

LE LIVRE DE L'AMSTRAD CPC464-CPC664

TOME 1

DANIEL MARTIN - PHILLIPE JADOUL



LE LIVRE DE L'AMSTRAD

D. Martin & Ph. Jadoul

**TOME I
SYSTEME DE BASE**

BCM 1985

AUTRES OUVRAGES EDITES CHEZ B.C.M.

Programmes internes du PET/CBM	par B. Michel
Le livre du VIC	par B. Michel
Le livre du 64	par B. Michel
Le livre du M.S.X.	par D. Martin
Les dessous du SPECTRAVIDEO	par D. Martin
Le livre du MS/PC-DOS	par F. Piette

MINIDISQUES MAGNETIQUES DISTRIBUES PAR B.C.M.

Le disque du 64	-	Le disque du MSX
Le disque de L'AMSTRAD	-	Le disque du PC/MSDOS

A PARAÎTRE :

Le livre de L'AMSTRAD tome 2 : Disques et périphériques.
Le livre de l'ATARI 520 ST

1ère édition Septembre 1985

Copyright B.C.M. s.c.

24, route de la Sapinière - 4960 BANNEUX - BELGIQUE
I.S.B.N. 2-87111-005-0 Dépôt légal D/1985/3827/6

Toute reproduction, non réservée à l'usage du copiste, d'un extrait quelconque de ce livre par quelque procédé que ce soit, est interdite sans l'autorisation écrite de l'éditeur.

DISTRIBUE PAR P.S.I. diffusion :

Place du Colonel Fabien , 5 - 75010 PARIS

I N T R O D U C T I O N

La machine AMSTRAD (SCHNEIDER pour les amis Belges) est devenue un BEST SELLER en l'espace de quelques mois, le modèle CPC464 à cassette s'est vendu par dizaine de milliers d'exemplaires et nous espérons que le CPC664 fera un aussi bon score.

L'AMSTRAD est certainement la machine qui offre le meilleur rapport qualité/prix pour l'année 1985.

Sa constitution interne que nous vous invitons à découvrir tout au long de ce manuel en fait une machine extensible dont la pérennité est assurée.

Nous espérons qu'à la lecture de ce livre, vous trouverez réponse à toutes vos questions et que vous l'utiliserez quotidiennement comme manuel de référence tant au point de vue matériel (HARDWARE) que logiciel (SOFTWARE).

Il ne vous reste plus qu'à en faire le meilleur usage pour vous décharger des aléas des techniques spécifiques et concentrer toute votre attention sur la logique de votre application.

Pour tirer un maximum de profit de la lecture de ce livre, de bonnes notions d'assembleur, une bonne connaissance du BASIC classique et une parfaite maîtrise de la notation hexadécimale sont indispensables.

Pour améliorer vos connaissances de l'assembleur et de la notation hexadécimale nous vous conseillons l'achat du livre : L'ASSEMBLEUR DE L'AMSTRAD par M. HENROT aux éditions P.S.I. En outre, l'ouvrage CLEFS POUR L'AMSTRAD de D. MARTIN, chez le même éditeur, constitue un manuel de référence indispensable à tout utilisateur avancé.

Fermez votre porte à clé, décrochez votre téléphone, envoyez votre conjoint en vacances et tournez la page pour découvrir le monde merveilleux de l'AMSTRAD.

Les auteurs

P. JADOUL & D. MARTIN

LES AUTEURS.

Philippe Jadoul : Ingénieur industriel en électronique, après des études à l'ISIL, il a suivi une formation complémentaire en technique des microprocesseurs au Centre de Recherche des Industries Fabrimétal (CRIF). Ingénieur de maintenance chez INTERTECHNIQUE pendant un an, il est depuis 1984 ingénieur au laboratoire de recherche de l'Institut National des Industries EXtractives (INIEEX).

Daniel Martin : Agrégé en mathématique-physique, après quelques mois au Ministère de l'éducation nationale, attiré par la micro-informatique, il entre comme COMPUTER MANAGER chez TANDY Corporation. Un bref passage chez APPLE aux Pays-Bas, et depuis 1981, il est ingénieur système chez INTERTECHNIQUE, le constructeur français spécialisé en mini-ordinateurs de gestion architecturés sur une base de données PICK. Fin 1984, il publie 'LE LIVRE DU MSX', début 85, 'LES DESSOUS DU SPECTRAVIDEO' et en août 1985, 'CLEFS POUR L'AMSTRAD'.

Une disquette au format 3 pouces reprenant tous les programmes du présent ouvrage est disponible chez B.C.M. et vendue au prix de 200 FF (1300 francs belges).

Ce livre à été composé sur un ordinateur IN50 de la société INTERTECHNIQUE sous système d'exploitation OASIS à l'aide du logiciel de traitement de texte MEDIATEXTE.

TABLE DES MATIERES

SECTION	PAGE
1	9
ORGANISATION INTERNE DE L'AMSTRAD.	
1.1	9
1.2	12
1.2.1	12
1.2.2	12
1.2.3	13
1.2.4	13
1.3	16
2	19
GESTION DE L'ECRAN : LE CRTC & LA VGA.	
2.1	19
2.2	23
2.2.1	23
2.2.2	25
2.2.3	27
2.2.4	30
2.3	32
2.3.1	32
2.3.2	32
2.3.3	35
2.3.4	36
2.4	37
2.4.1	37
2.4.2	38
2.4.3	41
2.5	43
2.5.1	43
2.5.2	44
2.5.3	45
2.5.4	46
2.5.5	47

3	LE GENERATEUR SONORE PSG AY3-8912	48
3.1	Généralités.	48
3.2	La théorie du son	49
3.2.1	Qualités du son	49
3.2.2	La hauteur	49
3.2.3	Le volume	51
3.2.4	Le timbre	51
3.2.5	Les bruits	51
3.2.6	Durée et attaque de la note	52
3.2.7	Paramètres définissant un son dans le CPC	52
3.3	Structure interne du PSG	54
3.3.1	Les différents registres du PSG	55
3.3.2	Les registres R0 à R5	56
3.3.3	Le registre R6	58
3.3.4	Le registre R7	59
3.3.5	Les registres R8 à R10	60
3.3.6	Les registres R11 et R12	61
3.3.7	Le registre R13	62
3.3.8	Les registres R14 et R15	63
3.4	Programmation des registres du PSG	64
3.5	Les routines internes du générateur sonore	70
4	L'INTERFACE PERIPHERIQUE PPI 8255A	73
4.1	Généralités	73
4.2	Découpage et utilisation des ports A,B & C	75
4.2.1	Introduction	75
4.2.2	Le port A	76
4.2.3	Le port B	76
4.2.4	Le port C	77
4.3	Programmation du PPI	79
4.3.1	Introduction	79
4.3.2	Programmation	79
4.4	Programmation du PSG au travers du PPI	83
4.5	Lecture du clavier au travers du PPI	87
4.5.1	Description du clavier	87
4.5.2	Lecture du clavier	88
4.6	Le joystick	91

5	LES PERIPHERIQUES EN CONTACT AVEC LE PPI	92
5.1	Le lecteur de cassette	92
5.1.1	Généralités	92
5.1.2	Structure générale d'un fichier	93
5.1.3	Format d'enregistrement	93
5.1.4	Format d'enregistrement des bits	95
5.1.5	Le HEADER	95
5.1.6	Vitesse d'écriture	97
5.1.7	Routines du gestionnaire cassette	98
5.2	L'interface imprimante Centronics	100
5.3	Le clavier	103
5.3.1	Fonctionnement	103
5.3.2	Les routines du gestionnaire clavier	106
6	STRUCTURE INTERNE DU BASIC AMSOFT.	107
6.1	Généralités	107
6.2	Composition des ROMs	108
6.3	Découpage de la mémoire centrale	109
6.4	Structure de la TIP	110
6.5	Structure de la table des variables (TV)	111
6.6	Structure de la table des chaînes (TC)	113
6.7	La région de communication	114
6.8	Fonctionnement du BASIC	115
6.9	Fonctions arithmétiques et mathématiques	117
6.10	La pile	118
6.11	Décomposition des ROMs	120
7	LES INSTRUCTIONS MAL CONNUES DU BASIC AMSOFT.	127
7.1	Généralités	127
7.2	Instruction DEF FN et fonction FN	128
7.3	Instruction MEMORY et fonctions HIMEM et FRE	131
7.4	Instruction POKE et fonction PEEK	134
7.5	Instructions OUT et WAIT et fonction INP	136
7.6	Instruction CALL	138
7.6.1	Introduction	138
7.6.2	Fonctionnement	139
7.7	Chargement d'un programme en langage machine	141
7.7.1	Méthode DATA et POKE	141
7.7.2	Méthode de la chaîne de caractère	142
7.7.3	Méthode de la variable tableau	144
7.8	Instructions SYMBOL et SYMBOL after	146
7.9	Instructions de gestion des interruptions	148

7.9.1	Etude théorique des interruptions	148
7.9.2	Instructions DI,EI,AFTER,EVERY et REMAIN	149
7.10	La fonction @ (VARPTR)	151
8	LES R.S.X.	159
8.1	Généralités.	159
8.2	Constitution d'un RSX	160
8.3	Programmation du traitement d'une commande	163
8.4	Programme BASIC de construction du RSX	168
8.5	Amélioration des RSXs	169
9	LE CPC 664	170
9.1	Introduction	170
9.2	Nouvelles fonctions et instructions du 664	171
9.3	Table des variables internes	173
9.4	Table des adresses ROM	176
9.5	Vecteurs mathématiques	179
9.6	Code et adresse d'exécution des mots clés	180
10.	PROGRAMMES, TRUCS ET ASTUCES.	182
10.1	Formattage d'un listing	183
10.2	Construction automatique de lignes DATA	185
10.3	Super Déplaceur	187
10.4	Super DUMP	191
10.5	Désassembleur en BASIC	193
10.6	RSX sur le SCROLLING	199
10.7	Addition vectorielle	202
10.8	RSX VECTID, VECINS et VECDEL	206
10.9	RSX PAINT	215
10.10	RSX RECPLEIN	222
10.11	RSX CERCLE	226
10.12	RSX SUPERSAUVÉ & SUPERCHARGE	235
10.13	RSX HARD COPY	245
	BIBLIOGRAPHIE	255

CHAPITRE I

ORGANISATION INTERNE ET STRUCTURE DE L'AMSTRAD.

1.1 Organisation interne.

Le schéma général de l'AMSTRAD CPC464 se trouve à la figure 1. Notez bien le sens des flèches qui relient les différents modules car il indique le sens du flux des données.

Le coeur du système est constitué par un microprocesseur Z80. Ce microprocesseur, originaire de chez INTEL, est un "vieux" de la micro-informatique. On le trouvait déjà, il y a 7 ans, dans le premier micro-ordinateur de chez TANDY, le TRS-80.

Le Z80 est doté de 16 fils d'adresse, qui lui permettent d'adresser 65536 cases mémoire différentes, et de 8 fils de données (raison pour laquelle le microprocesseur Z80 est dit 8 BITS), qui lui permettent de mémoriser un nombre compris entre 0 et 255 dans chaque case mémoire.

Dans l'architecture du CPC464, le Z80 contrôle 64 K (65536) octets de mémoire vive (RAM) et 32 K de mémoire morte (ROM). Dans le cas du CPC664, la ROM est portée à 48 K. Des explications complémentaires concernant la structure de la mémoire seront fournies dans la section 1.3.

Outre ses possibilités d'adressage mémoire, le Z80 possède la faculté d'adresser des PORTS d'entrée/sortie. Grâce à ces PORTS, le Z80 communique avec le contrôleur CRT 6845 et avec la VIDEO GATE ARRAY (VGA) qui se charge de la gestion de l'écran vidéo. Le chapitre 2 vous donnera toutes les explications quant au fonctionnement du CRT et de la VGA.

Les autres périphériques connectés aux ports d'entrée/sortie du Z80 sont le PPI (Parallèle Périphéral Interface) et le port de contrôle de l'imprimante.

Le PPI se charge du contrôle de la cassette, de l'adressage de la matrice clavier ainsi que de la gestion du contrôleur sonore PSG AY3-8912. Le chapitre 4 du présent volume vous révélera tout ce que vous avez toujours voulu savoir au sujet du PPI.

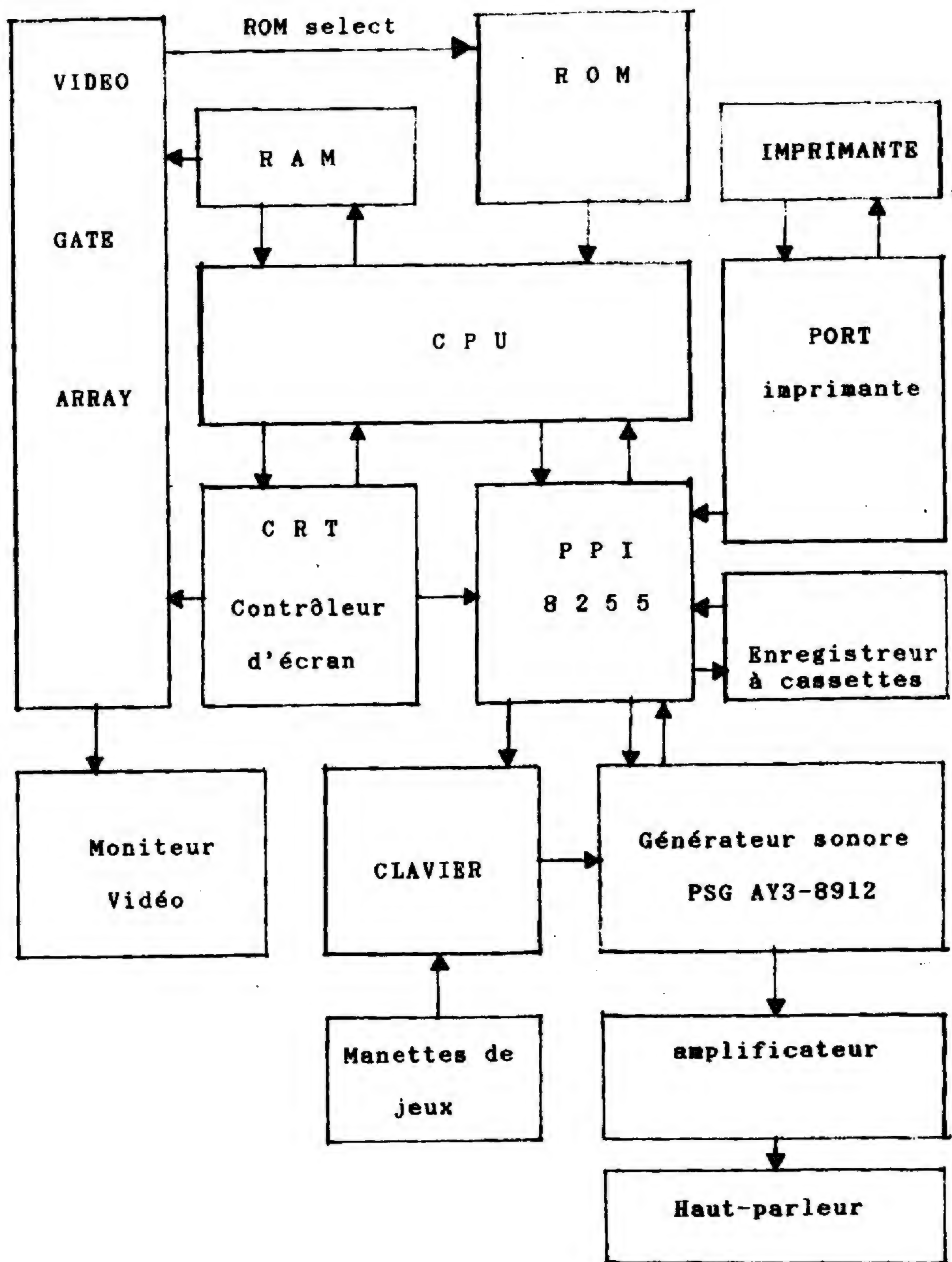
Enfin, le dernier bloc important est le générateur sonore. Il sera étudié en détails au cours du chapitre 3. Il n'est pas directement en ligne avec le microprocesseur mais est programmable au travers du PPI. Le PSG présente deux fonctions principales: la production du son et la lecture du clavier.

Les heureux possesseurs d'un CPC664 ou d'un lecteur de disque verront leur schéma bloc s'étoffer d'un contrôleur de disque de type NEC PD765A en contact direct avec le processeur Z80 par l'intermédiaire de ports spécialisés.

Ajoutons à cela quelques blocs mineurs comme l'alimentation, l'enregistreur à cassettes, l'amplificateur et le haut parleur et nous avons une excellente vue globale de l'AMSTRAD.

Le Z80 est piloté par un quartz qui oscille suivant une fréquence de 4 Mhz. Pour des raisons pratiques d'interfaçage avec le matériel, cette vitesse est réduite à 3,3 Mhz.

Comme vous avez pu le constater, le système est remarquablement économique au point de vue matériel, ce qui explique le prix d'achat très intéressant de la machine. Mais rassurez-vous, cette "braderie" matérielle est largement compensée par un système d'exploitation (logiciel) du haut niveau. Les chapitres 5 à 8 ont pour but de décrire les merveilleuses possibilités du logiciel de l'Amstrad.



1.2 Structure de la mémoire.

1.2.1 GENERALITES.

Examinons le diagramme de la figure 2. Notons tout d'abord que toutes les adresses sont représentées sous forme hexadécimale et ce, pour des raisons évidentes de commodité. Le présent manuel étant abondamment truffé d'adresses hexadécimales, autant s'y habituer tout de suite.

Comme nous avons vu (point 1.1) que le système de base adresse 64 K de RAM et 32 K de ROM et que par contre, nous avons vu que le Z80 ne peut adresser que 64 K, nous pouvons nous demander où se trouve l'astuce ?

Le système peut sélectionner des tranches de mémoire de 16 K appelées BANKS. Cette opération est réalisée par l'intermédiaire de la VGA (VIDEO GATE ARRAY) et nous l'étudierons en détails au cours du chapitre II.

Bien entendu, à chaque instant, seuls 64 K sont adressables. Ils peuvent être constitués de 64 K de RAM, ou de 48 K de RAM et de 16 K de ROM, ou de 32 K de RAM et de 32 K de ROM.

1.2.2. STRUCTURE DE LA ROM

La ROM est composée de 2 parties distinctes de 16K dans le CPC464.

La première occupe les adresses 0000 à 3FFF et sera appelée ROM BIOS, elle contient les routines principales de gestion de votre ordinateur ainsi que l'ensemble des fonctions mathématiques.

La seconde occupe les adresses de C000 à FFFF et sera appelée ROM BASIC. Comme son nom l'indique, elle contient les routines principales de traitement des mots clés du langage BASIC.

Le CPC664 quant à lui possède une troisième ROM de 16 K en parallèle sur la ROM BASIC (adresses C000 à FFFF). Cette ROM contient les primitives de gestion du disque ainsi

que les routines d'interfaçage du disque avec le matériel.

1.2.3. STRUCTURE DE LA RAM.

La mémoire RAM est constituée de 8 circuits de 64 K X 1 bit.

Analysons son contenu en détail:

- Les adresses de 00 à 3F contiennent la même chose que la ROM BIOS. Cette duplication est nécessaire pour permettre aux instructions RST (restart) du Z80 de trouver le code exécutable quel que soit le type de mémoire sélectionnée.
- A partir de l'adresse 40, les contenus de la ROM et de la RAM sont différents. La RAM contient des espaces de travail propres au langage utilisé (BASIC).
- La partie médiane de la RAM constitue l'espace de travail principal. L'espace compris entre 4000 et C000 est le seul à être toujours accessible directement. Il serait donc téméraire et particulièrement dangereux de positionner un pointeur de pile hors de cet espace.
- La partie supérieure de la RAM centrale contient les variables système. Vous trouverez plus de détails au sujet de cette partie en lisant le chapitre V et les suivants.
- L'espace supérieur suivant est constitué par les BLOCS DE SAUTS qui fournissent des points d'entrée standard pour la plupart des routines système importantes.
- De C000 à FFFF, la RAM est réservée à l'écran (voir chapitre II).

1.2.4. GESTION DE LA MEMOIRE.

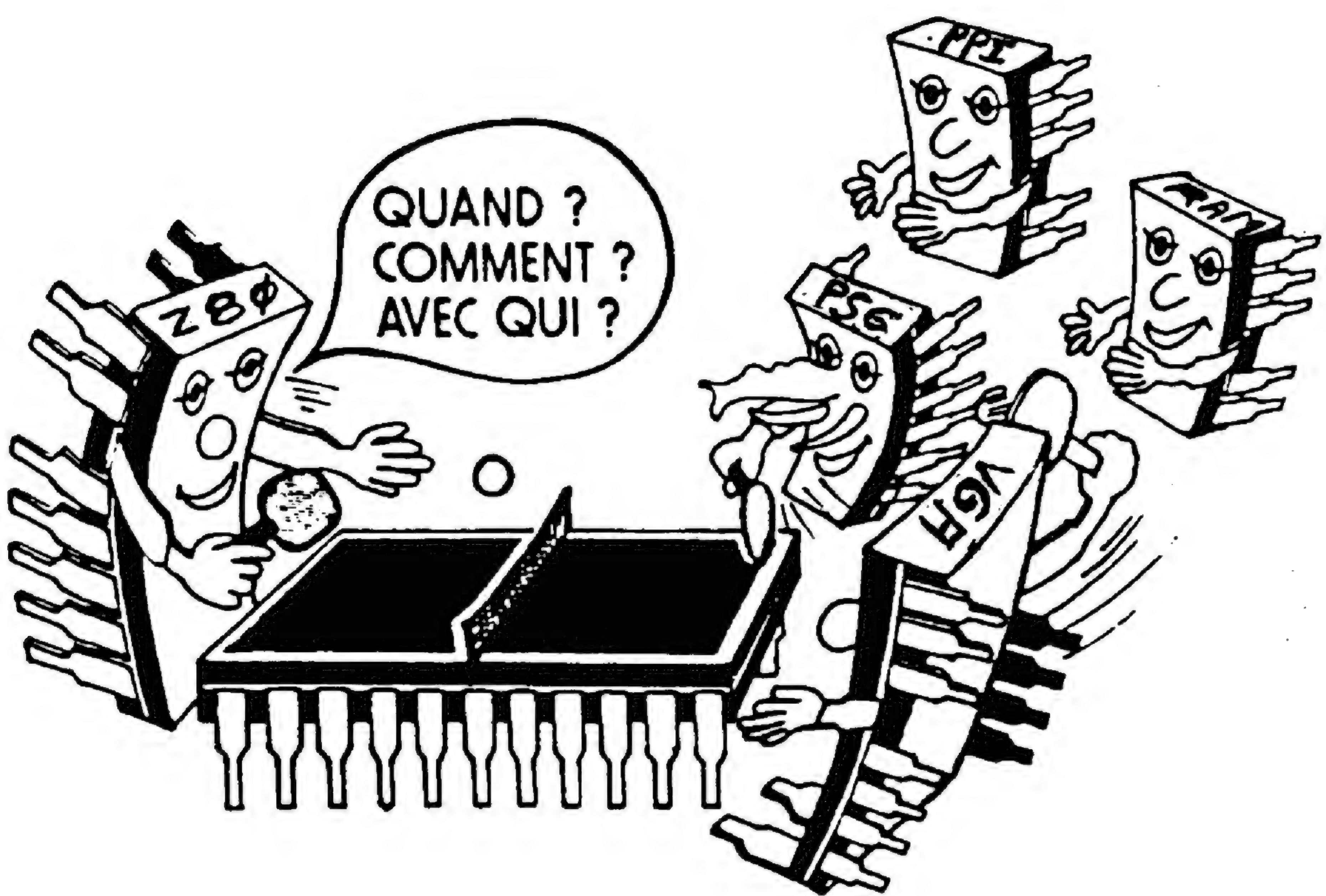
Toute écriture (présence du signal WR du processeur Z80) se produit dans la RAM. Il n'y a en effet aucun intérêt à écrire dans une ROM.

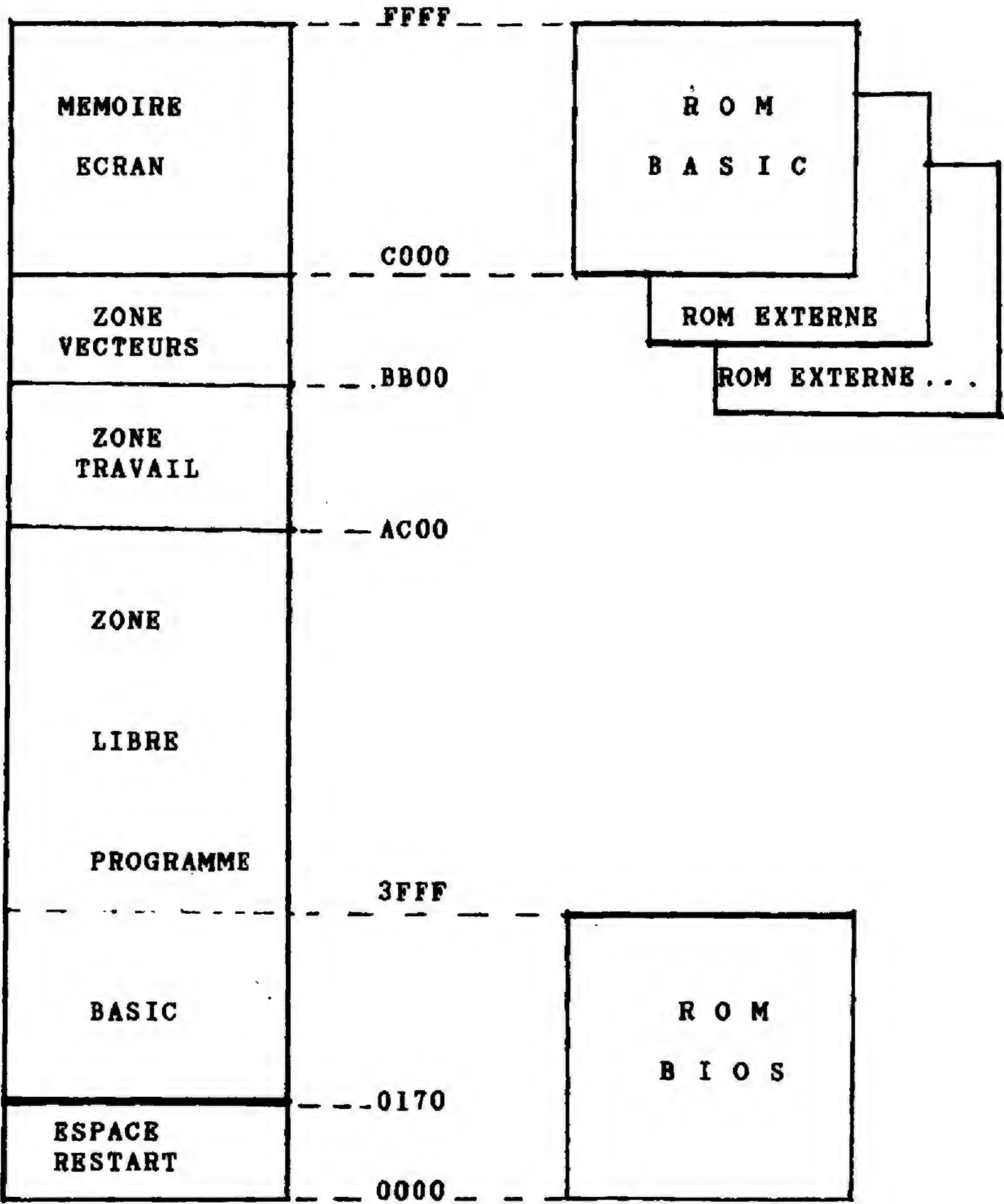
Si la ROM est hors circuit, la lecture se fera dans la RAM. Il faut noter que les deux blocs de la ROM sont indépendants l'un de l'autre. On peut en sélectionner soit un seul, soit les deux.

Les deux blocs constituant la ROM font en réalité partie intégrante d'un seul et même composant: une ROM de 32

K X 8 bits. Les lignes d'adresses sont structurées de la façon suivante:

Z80 A14	Z80 A15	ROM A14	PARTIE BIOS	PARTIE BASIC
0	0	0	SELECT	HORS CIRCUIT
1	1	1	H.C.	SELECT
0	1	X	H.C.	H.C.
1	0	X	H.C.	H.C.





MEMOIRE RAM

MEMOIRE ROM

1.3 Les Ports d'entrée/sortie.

La figure 3 nous fournit une table des différents ports d'entrée/sortie ainsi que leur utilisation.

Une première remarque apparaît à la lecture du tableau: les numéros des ports sont compris entre 0 et FFFF alors que le Z80 est bien connu pour avoir 256 (FF) ports d'entrée/sortie.

En réalité, le Z80 possède 65536 (FFFF) ports d'entrée/sortie.

En effet, lorsqu'on effectue une écriture ou une lecture (OUT, IN) sur le port dont le numéro est contenu dans le registre C du processeur, le registre B est copié sur les adresses de poids fort. Ainsi, un chargement préalable du registre B avec une valeur bien définie permettra une sélection plus fine des ports.

Exemple: pour écrire n sur le port FADC, il faut écrire:

```
LD    B, #FA
LD    C, #DC
OUT   (C), n
```

De cette structure, on peut déduire que les instructions INIR, INDR, OTIR et OTDR ne sont pas exécutables sur l'Amstrad car elles utilisent le registre B comme compteur.

L'arrangement des adresses des ports d'entrée/sortie peut sembler arbitraire et pourtant, en les examinant de plus près, nous pouvons remarquer que :

Tous les fils d'adresses se trouvant à l'état haut (1) :

- Si A15 est bas, la VGA est sélectionnée.
- Si A14 est bas, le CRT est sélectionné.
- Si A13 est bas, la ROM d'extension est sélectionnée.
- Si A12 est bas, l'imprimante est sélectionnée.
- Si A11 est bas, le PPI est sélectionné.
- Si A10 est bas, l'extension de bus est sélectionnée.

Un seul des bits compris entre A10 et A15 peut se trouver à l'état bas à un instant donné. Dans le cas

contraire, des conflits d'adressage se produiraient, ce qui explique le nombre important de ports interdits.

Dans les cas du CRT et du PPI, l'état des bits d'adresse A8 et A9 établissent le fonctionnement particulier du périphérique.

Dans les cas des ports F8 à FB (A10 bas), le bit A7 est à 0 si le FDC (contrôleur de disque) est utilisé; le bit A5 est à 0 si l'interface série est utilisée; le bit A6 est à 0 pour d'autres extensions.

A0 à A7 : ports F8 à FB.

00 à 7B : INTERDIT
7C à 7F : INTERFACE DISQUE
80 à BB : INTERDIT
BC à BF : RESERVE POUR USAGE ULTERIEUR
C0 à DB : INTERDIT
DC à DF : COMMUNICATION - RS 232
E0 à FE : RESERVE A L'UTILISATEUR
FF : RESET

Le port FFX est réservé à l'utilisateur. Autrement dit, à ses propres extensions.

F I G U R E 3.
Table des adresses et des fonctions des ports.

ADRESSE	SORTIE	ENTREE
00XX à 7EXX	interdit	interdit
7FXX	VGA	interdit
80XX à BBXX	interdit	interdit
BCXX	CRT sélection registre	interdit
BDXX	CRT donnée	interdit
BEXX	interdit	CRT état
BFXX	interdit	CRT donnée
COXX à DEXX	interdit	interdit
DFXX	sélection ROM externe	interdit
E0XX à EEXX	interdit	interdit
EFXX	port IMPRIMANTE	interdit
FOXX à F3XX	interdit	interdit
F4XX	PPI port A	PPI port A
F5XX	PPI port B	PPI port B
F6XX	PPI port C	PPI port C
F7XX	PPI contrôle	interdit
F8XX à FBXX	BUS D'EXTENSION	BUS D'EXTENSION
FA7E	FDC contrôle moteur	interdit
FADC	SIO données canal A	SIO données canal A
FADD	SIO contrôle canal A	SIO contrôle canal A
FADE	SIO données canal B	SIO données canal B
FADF	SIO contrôle canal B	SIO contrôle canal B
FB7E	interdit	FDC registre d'état
FB7F	FDC données	FDC données
FBDC	8253 compteur 0	8253 compteur 0
FBDD	8253 compteur 1	8253 compteur 1
FBDE	8253 compteur 2	8253 compteur 2
FBDF	8253 mode	8253 mode
FCXX à FEXX	interdit	interdit
FFXX	libre pour vos développements.	

GESTION DE L'ECRAN : LE CRTC ET LA VGA.

2.1 Généralités.

La gestion de l'écran du CPC464-664 a été confiée à deux circuits principaux. Le CRTC 6845 de Motorola et la VGA (VIDEO GATE ARRAY). Ce dernier circuit est spécialement fabriqué pour AMSTRAD et ne possède pas d'identification commerciale. Si le CRTC est complètement destiné à la gestion de l'écran, la VGA par contre est aussi utilisée pour réaliser la commutation des ROMs.

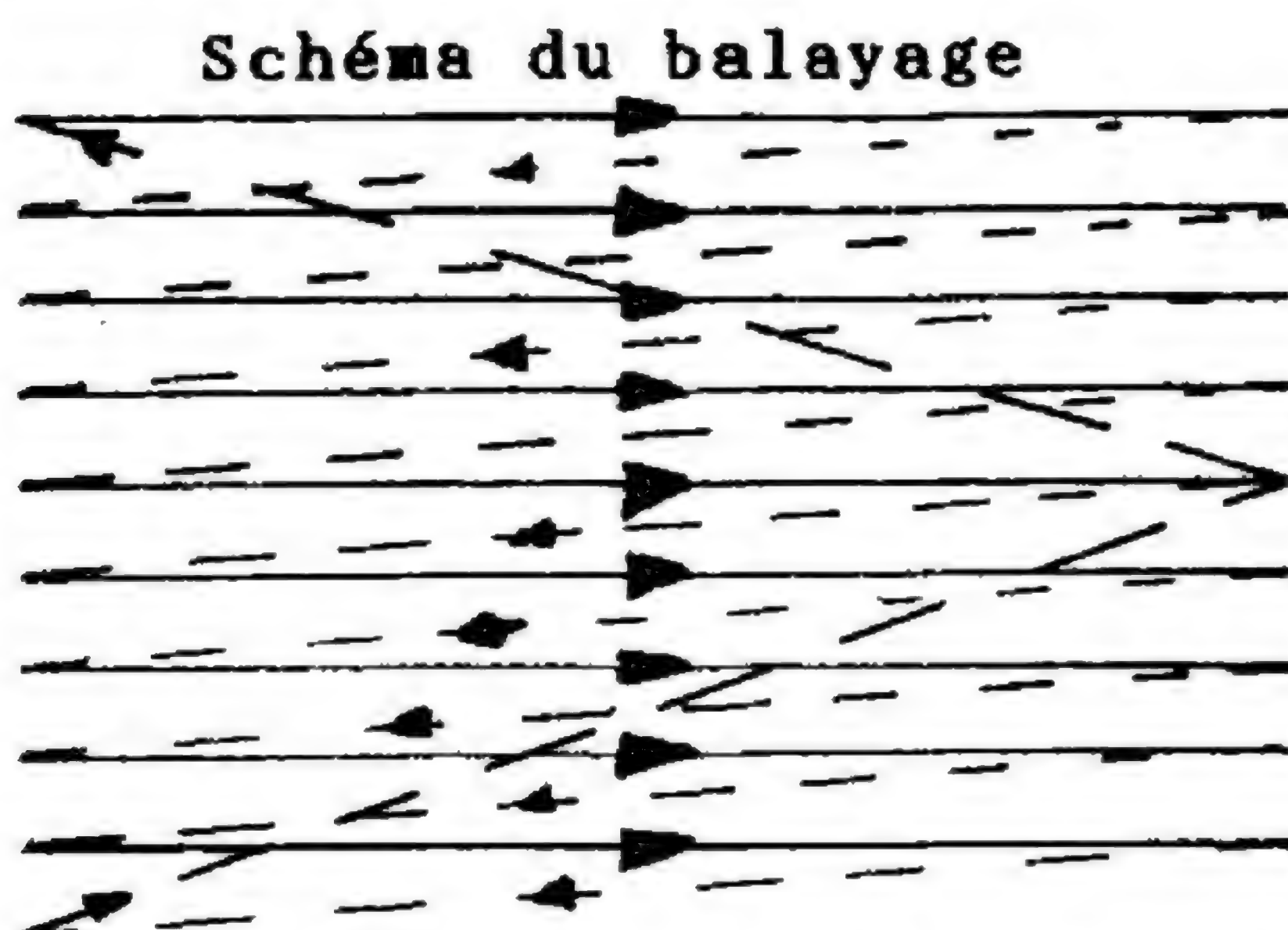
Cette section n'a pas pour but de vous fournir une explication détaillée sur la génération d'image vidéo, mais bien de définir les quelques concepts fondamentaux utiles à la compréhension de la suite de ce chapitre.

COMPOSITION : Un écran vidéo est composé principalement d'un tube cathodique, c'est un tube de verre dont la région frontale, appelée écran, est recouverte d'une matière fluorescente. Un canon à électrons positionné à l'arrière du tube émet un faisceau d'électrons qui frappe la surface fluorescente et produit ainsi un SPOT lumineux.

DEFLEXION : La position du SPOT peut être contrôlée par un dispositif électromagnétique appelé déflection. Il existe deux types de déflections, la déflection verticale et la déflection horizontale.

BALAYAGE : Le déplacement du spot se fait sur la surface de l'écran suivant un schéma bien précis. Le SPOT part du coin supérieur gauche de l'écran et se dirige suivant une horizontale vers le coin supérieur droit. Le retour à gauche se fait en diagonale en descendant d'une ligne. Arrivé en bas d'écran, la remontée a lieu en un temps beaucoup plus bref suivant une double diagonale (voir schéma). Pendant le trajet de retour et pendant le déplacement droite gauche, l'intensité du faisceau est considérablement diminuée, le balayage du spot "apparaît" alors comme une succession de

droites horizontales se déplaçant de gauche à droite.



PERSISTANCE : Durée de la trace d'un spot sur un point après le déplacement de celui-ci.

IMAGE : La variation d'intensité du spot pendant son parcours horizontal de gauche à droite produit une séquence de points allumés ou éteints. Cette succession produit une image. Exemple : pour produire un R à l'écran, la succession de points suivante doit être produite.

	1	2	3	4	5	6	7
1
2
3
4
5
6
7
8

L'intensité du faisceau est réduite sur la première ligne à l'issue de la colonne 5, sur la deuxième ligne pendant les colonnes 3 à 5

NOMBRE DE LIGNES VERTICALES : Dans nos régions, la fréquence du réseau est de 50 hertz. La fréquence du balayage interne d'un écran classique est de 15.625 hertz. On peut donc tracer $15625/50$ soit 312,5 lignes théoriques sur un écran. Cependant, un certain temps est utilisé pour le retour du balayage. Un vidéo normal demande le temps correspondant à une vingtaine de lignes pour effectuer son retour vertical. Le nombre de lignes actives dans le système CPC est fixé à 200. Ces lignes sont aussi appelées LIGNES TRAME.

NOMBRE DE LIGNES HORIZONTALES : Ce nombre dépend de la fréquence vidéo utilisée, ainsi pour tracer 640 points (la résolution du CPC), il vous faut, en tenant compte du retour horizontal (20 % supplémentaire soit 800 points) :

15625 x 800 = 12,5 Mégahertz.

ENTRELACEMENT : Cette méthode utilisée en télévision et possible avec le CRTC 6845 n'est pas utilisé dans le CPC. Elle consiste à afficher la totalité de l'écran en 2 passages, le premier sur les lignes impaires et le second sur les lignes paires. Dans ce cas, la résolution verticale théorique est doublée et passe à $312,5 \times 2 = 625$ lignes. On retrouve bien le standard télévision.

CRTC : Le CRTC est un dispositif à faible coût chargé de produire tous les signaux vidéo nécessaires à la production d'une image. Ces signaux sont au nombre de trois.

- La synchronisation horizontale (HSYNC) qui contrôle la déflexion horizontale.
- La synchronisation verticale (VSYNC) qui contrôle la déflexion verticale.
- Le signal vidéo qui contient l'information d'intensité pour chacun des points de l'écran.

VIDEO COMPOSITE : Signal composé des 3 précédents.

MATRICE DE CARACTERE : Rectangle dans lequel un caractère quelconque peut tenir. La matrice du CPC est un carré de 8 x 8 points.

GENERATEUR DE CARACTERES : Lors de la discussion sur l'image nous avons introduit le concept physique de production d'un caractère. Un caractère étant contenu dans une matrice de 8 x 8 points, il faut 64 bits pour définir un caractère complet. Un caractère étant codé en ASCII sur 8 bits, il faut un dispositif intermédiaire qui associe chaque code ASCII à une table de 8 octets (64 bits). Cette table est appelée générateur de caractères.

Cette table est contenue en ROM de l'adresse 3800 à l'adresse 3FFF soit 2048 octets qui permettent la définition de $2048 : 8 = 256$ caractères.

Remarque : l'instruction BASIC SYMBOL AFTER permet de recopier tout ou partie de la ROM génératrice de caractères en RAM autorisant ainsi la modification des matrices.

MEMOIRE ECRAN : Le CRTC dont la fonction consiste à présenter au dispositif d'affichage un signal correspondant aux points allumés ou éteints doit trouver l'information quelque part.

C'est ce rôle que joue la mémoire écran, elle contient l'information sur les points. Dans le CPC, c'est une partie de la mémoire centrale située entre C000 et FFFF qui joue ce rôle.

NIVEAU DE GRIS : Le dispositif décrit jusqu'à présent

affiche des points ou ne les affiche pas. Il est bien sûr possible d'afficher un point de façon plus ou moins lumineuse. Cette information (la luminosité du point) est appelée NIVEAU DE GRIS. L'information du niveau de gris doit évidemment être mise en mémoire. Si un bit suffit pour déterminer si un point est éteint ou allumé, il faut 4 bits par point si on veut afficher 16 niveau de gris (2^4).

REMARQUE : Le CPC étant un système couleur, les notions de base développées ici se compliquent passablement. Il n'entre pas dans notre intention de s'étendre sur la théorie de la production d'images couleurs et nous conseillons au lecteur désireux d'approfondir le sujet de consulter des manuels spécialisés. Pour simplifier le raisonnement, il suffit de considérer qu'un niveau de gris correspond à une couleur.

2.2 Le CRTC : étude technique.

2.2.1 COMPOSITION

Le CRTC 6845 est un circuit intégré LSI 40 broches qui génère tous les signaux nécessaires à la production d'une image vidéo.

PRINCIPAUX SIGNAUX :

- Un bus bidirectionnel 8 bits (D0-D7) pour transférer des données entre le processeur et ses registres internes.

Signaux entrants :

- Signal de sélection CS.
- Signal RW indiquant le sens de l'information (READ/WRITE).
- Signal E de synchronisation d'horloge.
- Signal CLK d'horloge.
- Signal RESET d'initialisation principale.
- Signal RS qui sélectionne le type de registre (Cf 2.2.2.).

Signaux sortants :

- 14 lignes d'adresses permettant la gestion de 16K de mémoire écran (MA0-MA13).
- 5 lignes de sélections pour le générateur de caractères.
- Signaux VIDEO , HSYNC et VSYNC.
- Signal LPTSB de gestion du photostyle (crayon lumineux).
- Signal CURSOR de validation du curseur.

SCHEMA FONCTIONNEL :

Le CRTC est composé de 7 blocs fonctionnels principaux :

- La logique d'adressage de mémoire écran.
- La logique de SCROLLING (déplacement d'une ligne).
- La logique de traitement du curseur.
- La logique de traitement du crayon optique.
- Le compteur de lignes.
- Le générateur de signaux de synchronisation.
- Le système de contrôle des registres.

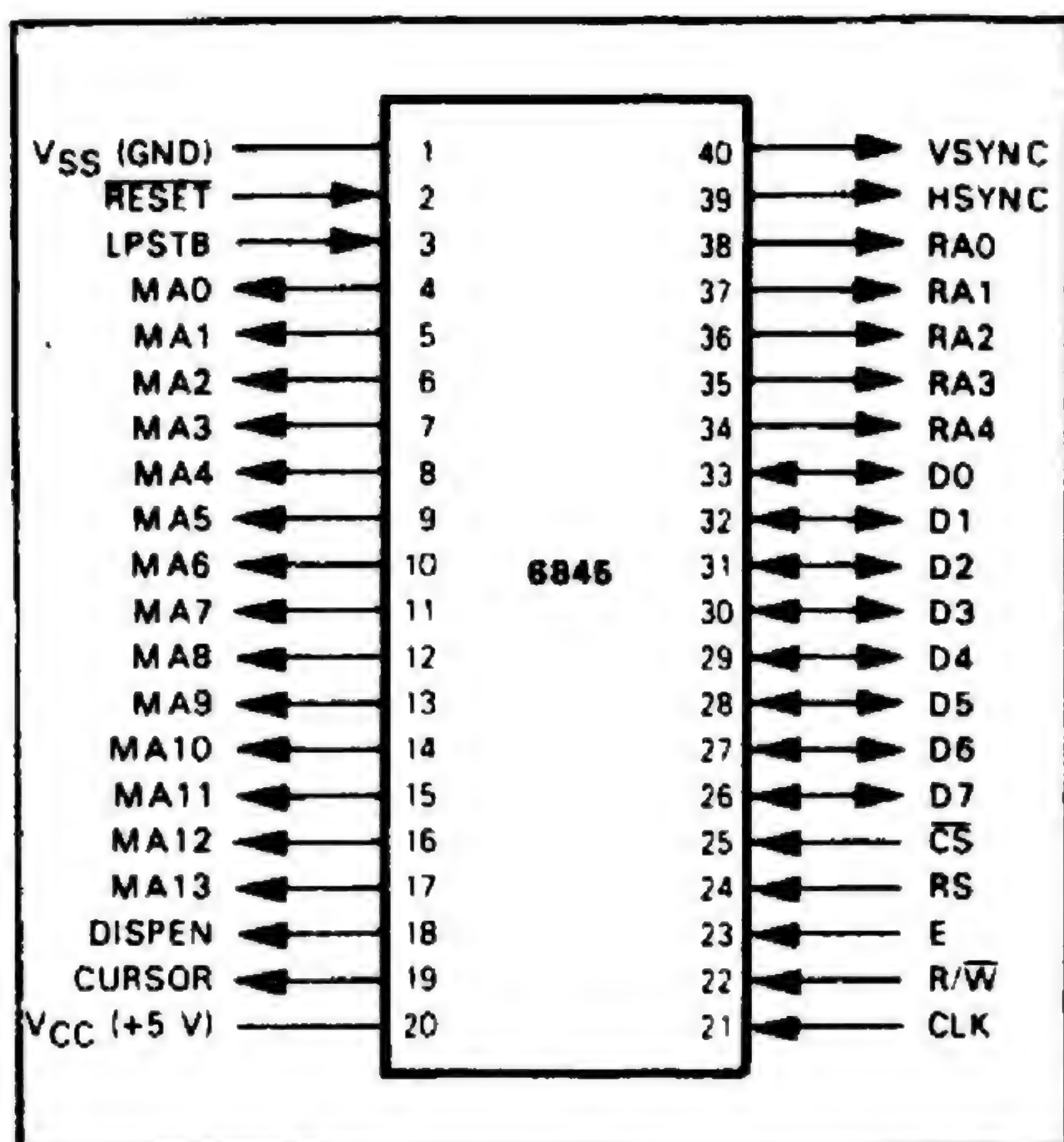
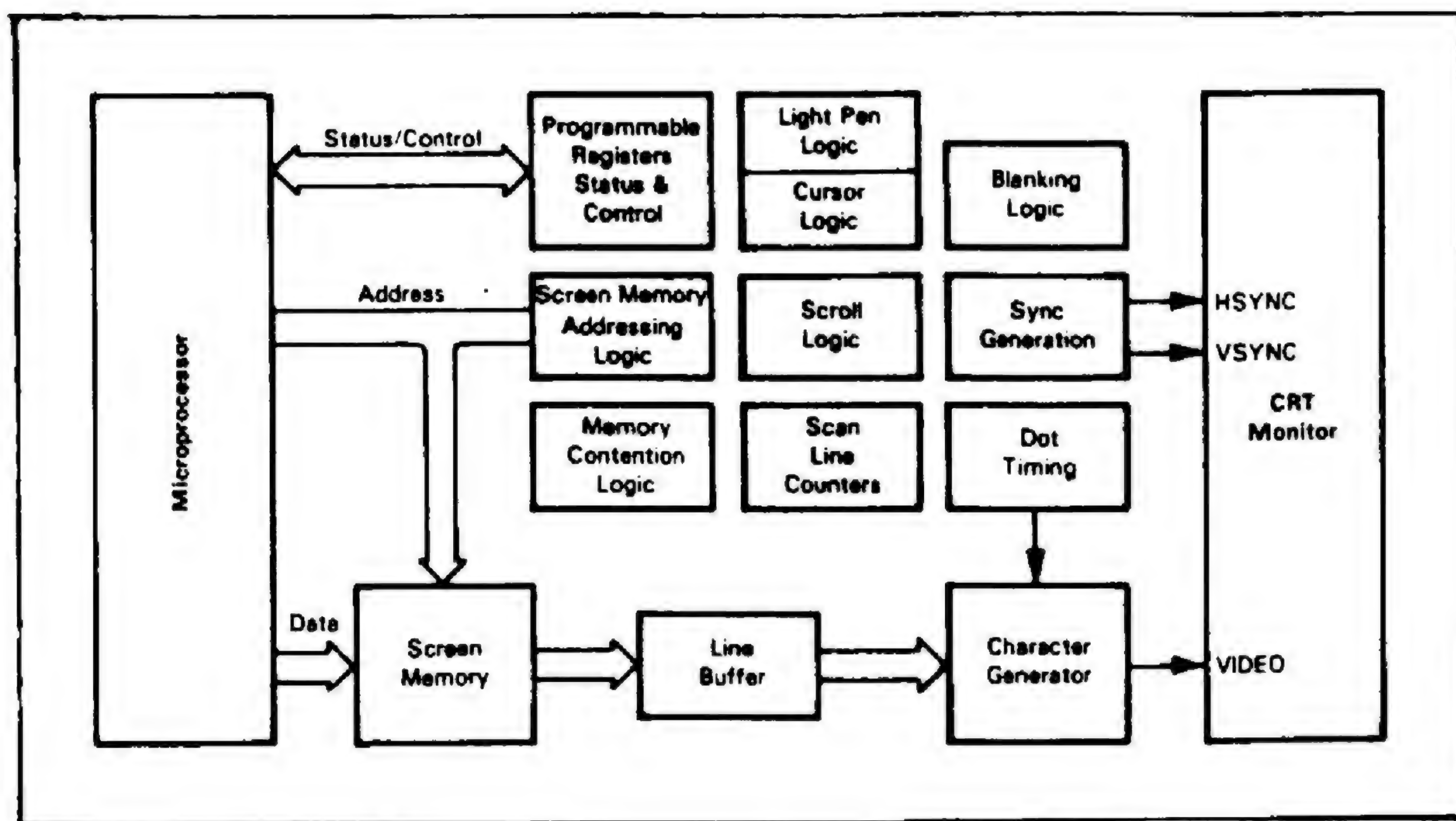


Schéma fonctionnel



2.2.2 LES REGISTRES PROGRAMMABLES

Le CRTC 6845 comporte 19 registres internes. Ces derniers doivent être accédés par le logiciel pour établir les paramètres de fonctionnement du circuit 6845. Un des registres est utilisé pour indiquer le registre sur lequel on travaille. Il est appelé REGISTRE ADRESSE. Les 18 autres sont les registres de paramètres.

L'accès à un registre de paramètres se fait en deux étapes.

- 1 : Fournir au REGISTRE ADRESSE le numéro du registre à accéder. Le nombre de registres à accéder étant de 18, 5 bits suffisent. Le registre adresse est donc un registre de 5 bits.
- 2 : Ecrire ou lire le registre spécifié.

Le signal RS introduit au point 2.3.2 détermine si le logiciel accède au registre adresse ou à un des autres registres. Cette technique permet d'économiser un certain nombre de broches sur le circuit intégré.

Les registres paramètres sont numérotés de R0 à R17. Ils peuvent être subdivisés en trois groupes principaux :

- 1: R0 à R3 : Programmation du format horizontal.
- 2: R4 à R9 : Programmation du format vertical.
- 3: R10 à R17: Contrôle du curseur, de la RAM et du crayon.

Les registres paramètres sont en général à écriture seule, à l'exception des deux registres de contrôle de la position du crayon qui sont à lecture seule et des deux registres de contrôle de la position du curseur qui sont utilisables aussi bien en écriture qu'en lecture.

La table de la page suivante décrit brièvement la fonction des 19 registres ainsi que leur nombre de bits et leur type de donnée.

Table des registres du CRTC 6845.

N°	FONCTION	L/E	BITS	VALEURS	TYPE
X	Registre adresse	E	5	0 à 17	REG
0	Total horizontal	E	8	1 à 256	CLK
1	Nombre de caractères par ligne	E	8	1 à 256	CLK
2	Position du signal HSYNC	E	8	1 à 256	CLK
3	Longueur du signal HSYNC	E	4	1 à 16	CLK
4	Total vertical en caractères	E	7	1 à 128	CAR
5	Ajustement du total vertical	E	5	1 à 32	LGN
6	Nombre de lignes réelles écran	E	7	1 à 128	CAR
7	Position du signal VSYNC	E	7	0 à 127	CAR
8	Mode entrelacé	E	2	0 à 3	
9	Nombre de lignes par écran	E	5	1 à 32	LGN
10a	Caractéristiques du curseur	E	2	0 à 3	
10b	Ligne de départ du curseur	E	5	1 à 32	LGN
11	Ligne de fin du curseur	E	5	1 à 32	LGN
12	Adresse haute début RAM	E	6	0 à 63	
13	Adresse basse début RAM	E	8	0 à 255	
14	Position du curseur BPS	L/E	6	0 à 63	
15	Position du curseur BMS	L/E	8	0 à 255	
16	Position du crayon BPS	L	6	0 à 63	
17	Position du crayon BMS	L	8	0 à 255	

E=ECRITURE, L=LECTURE, CLK=CLOCK, CAR=CARACTERE, LGN=LIGNE

2.2.3 ETUDE DETAILLEE DES REGISTRES

A) Les registres de programmation du format horizontal.

Ces registres sont chargés avec une valeur bien établie lors de l'initialisation du CRTC et ne doivent en principe pas être modifiés.

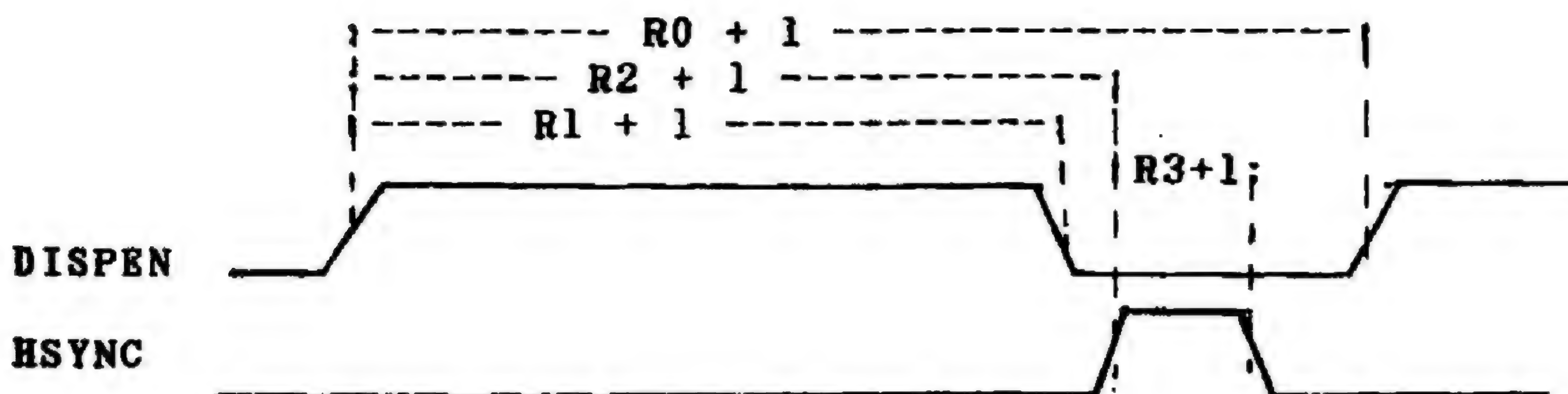
R0 : Contient le temps total d'une ligne horizontale y compris le temps de retour du spot. Attention, l'unité de temps est le temps d'un caractère. On peut donc dire que R0 contient le nombre de caractères affichables sur une horizontale y compris les caractères affichés pendant le retour du spot. La valeur de R0 peut être comprise entre 0 et 255 (8 bits) mais le nombre de caractères est supérieur d'une unité à cette valeur. R0 contient donc un nombre de caractères compris entre 1 et 256.

R1 : Contient le nombre de caractères réellement affichables. Ce nombre suit la même règle que celui de R0, le nombre de caractères affichables est donc compris entre 1 et 256.

R2 : Contient le nombre de caractères après lequel le signal de synchronisation horizontale HSYNC change d'état. Ce nombre est toujours supérieur au nombre contenu dans R1. IL suit la même règle que les deux précédents.

R3 : Contient la durée du signal HSYNC exprimée en nombre de caractères. Ce nombre est exprimé sur 4 bits (0 à 15) et contient le nombre de caractères moins 1. Le nombre de caractères du signal est donc compris entre 1 et 16.

Schéma de l'activité des registres R0 à R3.



B) Les registres de programmation du format vertical.

Remarque : Dans cette partie, établissez bien la différence entre une ligne de caractères au sens commun (80 caractères par ligne) et une ligne de trame (voir lignes verticales au point 2.3.1).

Comme les précédents, ces registres sont chargés avec une valeur précise et ne doivent pas être modifiés.

R4 : Contient le nombre de caractères affichables verticalement - 1. Il comprend les caractères de retour du spot. Ce nombre est forcément plus grand que celui représentant le nombre de caractères réellement affichés sur l'écran. Le nombre de lignes de trame se déduit d'après la taille en nombre de lignes de trame d'un caractère. R4 ne permet donc pas un réglage en nombre de lignes de trame très fin. R4 est exprimé sur 7 bits (0 à 127).

R5 : Contient le nombre de lignes de trame additionnelles à ajouter à R4, R5 joue le rôle de réglage fin. R5 est un registre de 5 bits (0 à 31) car la taille d'un caractère en nombre de lignes de trame peut être comprise entre 0 et 31.

R6 : Contient le nombre de caractères réellement affichés sur l'écran en vertical.

R7 : Contient le nombre de lignes de trame après lequel le signal VSYNC apparaît. Ce signal dure le temps de 16 lignes de trame.

R8 : Ce registre de 2 bits contient le mode du CRT. Les valeurs 0 et 2 correspondent au mode normal, les valeurs 1 et 3 ne peuvent être utilisées, elles correspondent au mode entrelacé.

R9 : Contient le nombre de lignes par caractère - 1 . Si le caractère est dans une matrice 8x8, ce registre contient 7.

C) Les autres registres.

R10 : Ce registre présente deux fonctions différentes. Les bits 6 et 5 déterminent l'aspect du curseur. Les bits 0 à 4 fournissent le numéro de la ligne à laquelle commence le curseur (0 à 31).

BIT 6	BIT 5	ASPECT
0	0	CURSEUR FIXE
0	1	PAS DE CURSEUR
1	0	CURSEUR CLIGNOTANT VITESSE RAPIDE
1	1	CURSEUR CLIGNOTANT VITESSE LENTE

R11 : Contient le numéro de la ligne à laquelle finit le curseur. Comme pour R10, il s'agit bien sûr de lignes trame.

R12 : Adresse haute du départ de la mémoire écran (6 bits).

R13 : Adresse basse du départ de la mémoire écran (8 bits).

Les registres R12 et R13 sont les 2 registres les plus couramment accédés dans le CPC.

R14 : Adresse haute de la position du curseur (6 bits).

R15 : Adresse basse de la position du curseur (8 bits).

R16 : Adresse haute fournie par le crayon optique lorsqu'on active celui-ci. Cette option n'est pas standard sur le CPC. Ce registre est aussi de 6 bits.

R17 : Adresse basse du registre du crayon optique (8 bits).

En plus des 18 registres décrits, le 6845 dans sa version S possède un registre supplémentaire appelé **REGISTRE STATUT**. Ce registre, utilisable en lecture uniquement, ne possède que 3 bits actifs (B7, B6 et B5).

B7 : 1 si le CRTC est accessible.

B6 : 0 si R16 et R17 ont été lus par le processeur, 1 si le signal du crayon a été détecté.

B5 : Indicateur de BLANKING . 0 si le spot est en phase d'affichage, 1 si le spot est en retour vertical.

Ce registre étant de peu d'utilité en dehors de l'utilisation du crayon optique, il ne sera plus considéré par la suite.

Il est inutile de préciser que tous les registres sont liés entre eux et que toute tentative de modification de l'un d'entre eux risque de détruire l'affichage écran et d'obliger le malheureux apprenti-sorcier à procéder à l'extinction de l'ordinateur. Chaque modification doit donc se faire après mûre réflexion et en ne perdant pas de vue que la VGA modifie elle aussi les registres du CRTC.

Valeur par défaut des registres à l'initialisation

Lors de l'initialisation (allumage) de l'AMSTRAD, les registres sont chargés avec une valeur de départ. Les registres R0 à R9 ne sont plus modifiés par la suite. Les registres R10 à R17 peuvent être modifiés en cours d'exploitation.

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	
63	40	46	142	38	0	25	30	0	7	Décimal
3F	28	2E	8E	26	0	19	1E	0	7	Hexa

2.2.4 PROGRAMMATION DIRECTE DU CRTC

Le circuit 6845 est programmable directement à travers 4 ports d'entrée/sortie situés aux adresses BCXX, BDXX, BEXX et BFXX.

A - BCXX : Ce port est utilisé en écriture uniquement, il sert à positionner le numéro du registre à atteindre dans le REGISTRE ADRESSE.

Exemple : Pour sélectionner le registre R8 en BASIC, faire :

```
10 OUT &BC00,8
```

B - BDXX : Ce port est utilisé en écriture uniquement, il sert à positionner une valeur dans un registre préalablement sélectionné au moyen du port précédent.

Exemple : Pour écrire 6 dans le registre R9, faire :

```
10 OUT &BC00,9 : OUT &BD00,6
```


C - BEXX : Ce port est utilisé en lecture uniquement, il sert à lire le REGISTRE STATUT du CRTC.

D - BFXX : Ce port est utilisé en lecture uniquement, il sert à lire le contenu d'un registre préalablement sélectionné par le port BCXX. Rappel : Seuls les registres 14,15,16 et 17 peuvent être lus (curseur et crayon).

EXPERIMENTATION :

Encodez le programme BASIC suivant, et essayez diverses valeurs pour R et pour V. Notez les effets obtenus.

!! *** Certaines valeurs peuvent "planter" le système *** !!

```
10 INPUT "NUMERO DU REGISTRE ";R
20 INPUT "VALEUR A ECRIRE ";V
30 PRINT
40 OUT &BC00,R : OUT &BD00,V
50 GOTO 10
```

QUELQUES EXEMPLES :

R	V	E F F E T S
0	62	Recul écran à gauche
0	64	Avance écran à droite
0	63	Retour normal
1	39	Ecran en diagonale
1	40	Retour normal
3	1	L'image tourne en horizontal
3	14	Retour normal
4	35	L'écran se serre vers le haut
4	38	Retour normal
8	1	Vibration de l'écran (entrelacement)
8	3	Image découpée.
8	0	Retour normal
9	6	Modification matrice caractère
9	8	Idem
9	7	Retour normal
12	X	Modification adresse mémoire (a faire après un CLS)

2.3 La VGA : étude technique.

2.3.1 DESCRIPTION

La VGA est un circuit hybride qui renferme plusieurs portes logiques élémentaires dans une configuration spécialement étudiée pour votre ordinateur. Ce composant spécial ne sera donc décrit qu'au travers de ses fonctions.

La VGA a pour fonctions de :

- 1 - Permettre la commutation des ROMs.
- 2 - Sélectionner le mode écran (résolution).
- 3 - Sélectionner l'encre dont on veut définir la couleur.
- 4 - Définir la couleur d'encre parmi une palette de 27.
- 5 - Réinitialiser le compteur d'interruption.

Pour réaliser ces différentes fonctions, la VGA est interfacée avec le processeur par l'intermédiaire du port 7FXX, comme nous l'avons vu à la section 1.3. Etudions en détail la structure de ce port.

2.3.2 STRUCTURE DU PORT DE COMMANDE DE LA VGA

Le port 7FXX, comme tous les autres, est un port de 8 bits. Les bits 7 et 6 (Rappel : les bits sont numérotés de 0 à 7 en partant de la droite, le bit 7 est donc le bit le plus significatif.) sont utilisés pour définir le type d'opération réalisée par la VGA.

BIT 7	BIT 6	- - - - - OPERATION - - - - -
0	0	Sélection de l'encre.
0	1	Sélection de la couleur d'encre désignée.
1	0	Sélection du mode, des ROMs...
1	1	Non utilisé.

Pour plus de facilité, chaque combinaison des bits 7 et 6 peut être considérée comme un registre particulier. La VGA est donc composée de 3 registres importants.

A) Registre de sélection de l'encre :

Bit 7 : 0
Bit 6 : 0
Bit 5 : 0 (inutilisé)
Bit 4 : Sélection encre (0) ou bord (1)
Bit 3 à 0 : numéro de l'encre.

Pour sélectionner une encre, il suffit de mettre les 4 bits de poids fort à 0 et de fournir le numéro de l'encre (0 à 15) sur les 4 bits de poids faible.

Exemple : Pour sélectionner l'encre 12, il suffit d'écrire 12 (décimal) sur le port F7XX : 00001100

Si le bit 4 est à l'état 1, les bits 0 à 3 ne sont pas pris en compte et c'est le bord de l'écran qui est désigné (BORDER).

Remarque : comme signalé dans les généralités, il y a une association entre les niveaux de gris générés par le CRTC et la couleur. Le système est capable de générer 16 niveaux de gris, la VGA se charge d'associer chacun de ces niveaux de gris à une couleur précise.

B) Registre de sélection de la couleur de l'encre.

Bit 7 : 0

Bit 6 : 1

Bit 5 : 0 (inutilisé)

Bit 4 à 0 : Sélection de la couleur (32 possibilités fournissant 27 teintes).

Le bord ou l'encre dont le numéro a été sélectionné au moyen du registre précédent (A) est "chargé" avec la couleur définie par l'état des bits 0 à 4 du présent registre.

Voici la table des couleurs en fonction des bits 0 à 4.

B4	B3	B2	B1	B0	VALEUR	COULEUR
0	0	0	0	0	0	BLANC
0	0	0	0	1	1	INUTILISE
0	0	0	1	0	2	VERT MARIN
0	0	0	1	1	3	JAUNE PASTEL
0	0	1	0	0	4	BLEU
0	0	1	0	1	5	POURPRE
0	0	1	1	0	6	TURQUOISE
0	0	1	1	1	7	ROSE
0	1	0	0	0	8	INUTILISE
0	1	0	0	1	9	INUTILISE
0	1	0	1	0	10	JAUNE VIF
0	1	0	1	1	11	BLANC BRILLIANT
0	1	1	0	0	12	ROUGE VIF
0	1	1	0	1	13	MAGENTA VIF
0	1	1	1	0	14	ORANGE
0	1	1	1	1	15	MAGENTA PASTEL
1	0	0	0	0	16	INUTILISE
1	0	0	0	1	17	INUTILISE
1	0	0	1	0	18	VERT VIF
1	0	0	1	1	19	TURQUOISE VIF
1	0	1	0	0	20	NOIR
1	0	1	0	1	21	BLEU VIF
1	0	1	1	0	22	VERT
1	0	1	1	1	23	BLEU CIEL
1	1	0	0	0	24	MAGENTA
1	1	0	0	1	25	VERT PASTEL
1	1	0	1	0	26	VERT CITRON
1	1	0	1	1	27	TURQUOISE PASTEL
1	1	1	0	0	28	ROUGE
1	1	1	0	1	29	MAUVE
1	1	1	1	0	30	JAUNE
1	1	1	1	1	31	BLEU PASTEL

C) Registre de sélection du mode, des ROMs ...

Bit 7 : 1
Bit 6 : 0
Bit 5 : 0 (inutilisé)
Bit 4 : Réinitialisation du compteur d'interruption.
Bit 3 : Sélection de la ROM supérieure.
Bit 2 : Sélection de la ROM inférieure.
Bit 1 et 0 : Sélection du mode

Bit 1	Bit 0	Mode
0	0	0 : 160 x 200 en 16 couleurs
0	1	1 : 320 x 200 en 4 couleurs
1	0	2 : 640 x 200 en 2 couleurs
1	1	x : ne pas utiliser

La mise à 1 du bit 4 remet à 0 le bit supérieur du diviseur qui génère les interruptions.

La mise à 0 des bits 3 et 2 sélectionne la ROM correspondante. Si ces bits sont à l'état 1, les ROMs sont hors circuit.

2.3.3 SYNTHÈSE.

La principale fonction de la VGA est d'obtenir les données en provenance de la mémoire écran sur la base des adresses fournies par le CRT et de les traiter pour fournir un signal exploitable pour le moniteur video.

En outre, les 3 registres sont à écriture seule. Autrement dit, on ne peut pas lire l'état courant du registre et déterminer la valeur courante d'une encre ou le mode écran. De plus, lors du changement de mode ou lors de la commutation d'une ROM, il faut laisser les autres bits dans leur état précédent. Il est donc indispensable de garder en mémoire centrale la dernière valeur envoyée sur un registre.

L'utilisateur n'a en général aucun intérêt à accéder à la VGA directement. Le système possède une multitude de routines qui en simplifient l'accès. Les principales routines seront décrites à la fin de ce chapitre. Cependant, si un temps de réponse extrêmement rapide est nécessaire, un accès direct est toujours possible et vous possédez maintenant tous les éléments nécessaires pour le réaliser.

2.3.4. PROGRAMMATION DIRECTE DE LA VGA

La VGA est programmable directement à travers le port 7FXX du processeur.

Il est très dangereux de sélectionner les ROMS en cours de programme BASIC aussi nous limiterons nous à la programmation des couleurs et des encres.

Exemple 1 : Programmation directe de l'encre 1 en rouge.

Une simple consultation de la table des couleurs décrite page 34 permet de déterminer les 5 bits associés à la couleur rouge (28) : 1 1 1 0 0.

Etat du registre de sélection de l'encre 1 :

0 0 0 0 0 0 0 1. soit 1 en décimal.

Etat du registre de sélection de la couleur rouge :

0 1 0 1 1 1 0 0. soit 92 en décimal.

Ecriture en BASIC : OUT &7F00,1 : OUT &7F00,92

Exemple 2 : Programmation directe du bord en noir.

Etat du registre de sélection du bord :

0 0 0 1 0 0 0 0 soit 16 en décimal.

Etat du registre de sélection de la couleur noire :

0 1 0 1 0 1 0 0 soit 84 en décimal.

Ecriture en BASIC : 10 OUT &7F00,16 : OUT &7F00,84

2.4 La mémoire écran.

2.4.1 GENERALITES

La mémoire écran, dont nous avons déjà brièvement parlé, occupe en standard à l'initialisation les adresses RAM comprises entre C000 et FFFF soit 16384 octets. Cette mémoire est donc en connexion directe avec le processeur Z80 et avec le CRTC.

La mémoire écran, située en C000, partage la même zone que la ROM supérieure (BASIC).

Il est possible de déplacer le bloc mémoire écran par pas de 4000 (hexa) et ainsi de l'installer aux adresses 0000, 4000, 8000 ou C000. Au vu de la structure de la mémoire centrale et des pointeurs, en dehors de C000, seul 4000 est une adresse acceptable pour son installation. En 0000, elle écrase les RSTs et en 8000, elle écrase les vecteurs système (voir chapitre 6).

Les modes d'affichage écran, au nombre de trois, sont numérotés de 0 à 2. Chaque mode a sa propre résolution et son propre nombre de couleurs. Les trois modes ont une résolution verticale identique de 200 lignes. La résolution horizontale est respectivement de 160, 320 et 640 pixels pour les modes 0, 1 et 2. Les pixels ont une taille respective de 4, 2 ou 1 points écran.

Un caractère est toujours constitué de 8 pixels horizontaux sur 8 lignes, le nombre de caractères affichables est donc de 20, 40 ou 80.

L'arrangement de la mémoire écran dépend du mode d'affichage choisi. Cependant, dans tous les modes, la mémoire écran peut être considérée comme 8 K mots de 16 bits. Chaque mot contenant 4, 8 ou 16 points dont le niveau de gris (la couleur) est définie sur 4, 2 ou 1 bits suivant le mode.

MODE	Nombre de point	Nombre de points/pixel	Nombre de caractères	Nombre de bits	Nombre de couleurs
0	4	4	20	4	16
1	8	2	40	2	4
2	16	1	80	1	2

2.4.2 STRUCTURE DE LA MEMOIRE ECRAN

La structure de la mémoire écran n'est pas simple, heureusement, une multitude de routines système existe pour faciliter la tâche du programmeur. Ces routines seront envisagées dans la section suivante. Cependant, certaines applications demandent un temps de réponse très bref (jeux d'arcades, animation...). Dans ce cas, il peut être indispensable d'accéder à la mémoire écran directement. Une étude complète de sa structure est donc nécessaire.

En considérant la mémoire écran composée de mots de 16 bits, elle est donc composée de 8 K mots de 16 bits.

Les 8 K mots sont divisés en 8 blocs de 1 K mots. Le premier va de C000 à C7FF et le dernier de F800 à FFFF.

Les 200 lignes verticales de l'écran étant numérotées de 0 à 199, la structure est la suivante :

Les données des lignes 0,8,16,24,...,192 sont contenues dans le premier K mots. Les données pour les lignes 1,9,17,...,193 sont contenues dans le K mots suivant. Les données pour les lignes 7,15,23,...,199 sont contenues dans le huitième et dernier K mots.

De cette structure, on peut déduire que chaque K mots contient les informations de 25 lignes numérotées de 8 en 8. Chaque ligne est composée de 40 mots de 16 bits. Il y a donc 1000 mots utilisés par K mots et 24 mots inutilisés (48 octets).

Comme chaque ligne est composée de 40 mots de 16 bits, chaque mot consécutif représente : 16 points ou 2 caractères en mode 2, 8 points ou 1 caractère en mode 1 et 4 points ou 1 demi caractère en mode 0.

Le diagramme suivant illustre bien la structure de la mémoire écran.

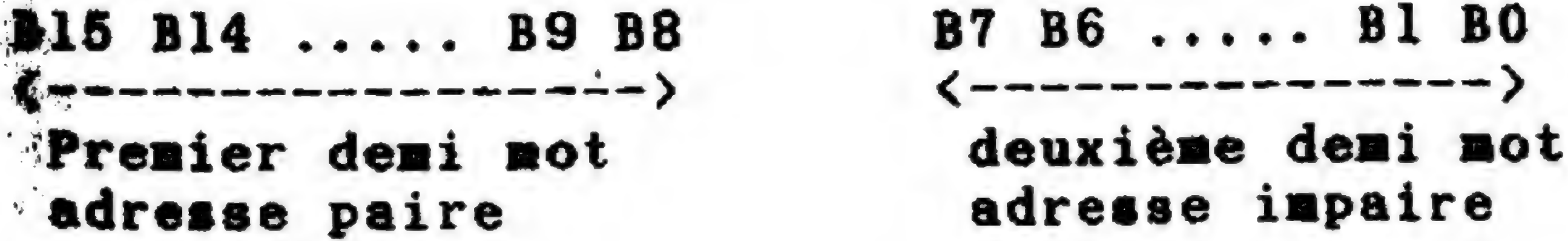
< - - - - - 40 MOTS DE 16 BITS - - - - - >									
:	C000	C002	C04C	C04E	:			
2	C800	C802	C84C	C84E	:	1		
0					:	C		
0					:	A		
	F000	F002	F04C	F04E	:	R		
L	F800	F802	F84C	F84E	:			
I	C050	C052	C09C	C09E	:	2		
G	C850	C852	C89C	C89E	:	C		
N					:	A		
E					:	R		
S					:			
:	FF80	FF82	FFCC	FFCE	:	25	CAR	

Remarque : La première ligne de caractères occupe les adresses C000 à C04F , C800 à C84F , jusqu'à F800 à F84F. La ligne de caractères suivante commence à l'adresse C050 et se termine en F89F. La dernière ligne de caractères commence en C780 et se termine en FFCF. Les adresses C7D0 à C7FF, CFD0 à CFFF ... FFD0 à FFFF ne sont pas utilisées.

Une modification d'OFFSET de début d'écran de 80 octets (50 hexa) dans le sens négatif ou positif produira un SCROLLING de l'écran vers le bas ou vers le haut. Cet effet est utilisé par le gestionnaire écran pour simuler le défilement de l'écran sur la 25 ième ligne.

Pour avoir une vue complète du problème, il reste à étudier la structure d'un mot en fonction du mode.

Les bits étant numérotés de B0 à B15 dans le mot en partant de la gauche.



- En mode 0 : 4 points de 4 bits.
Le point le plus à gauche est identifié par les 4 bits B9, B13, B11 et B15 dans l'ordre. Le point suivant par les bits B8, B12, B10 et B14. Le troisième point par B1, B5, B3 et B7

et le dernier par B0, B4, B2 et B6.

- En mode 1 : 8 points de 2 bits.

point 1 : B11 et B15	-	point 2 : B10 et B14
point 3 : B9 et B13	-	point 4 : B8 et B12
point 5 : B3 et B7	-	point 6 : B2 et B6
point 7 : B1 et B5	-	point 8 : B0 et B4

- En mode 2 : 16 points de 1 bit.

Le point le plus à gauche est déterminé par B15 et le point le plus à droite est déterminé par B0.

Les bits étant numérotés de gauche à droite de 15 à 0, les points étant numérotés de gauche à droite de P0 à P15 et les bits de couleurs étant numérotés de gauche à droite BC3 à BC0, le diagramme suivant apparaît :

BIT	MODE 0	MODE 1	MODE 2
15	P0 BC0	P0 BC0	P0
14	P1 BC0	P1 BC0	P1
13	P0 BC2	P2 BC0	P2
12	P1 BC2	P3 BC0	P3
11	P0 BC1	P0 BC1	P4
10	P1 BC1	P1 BC1	P5
09	P0 BC3	P2 BC1	P6
08	P1 BC3	P3 BC1	P7
07	P2 BC0	P4 BC0	P8
06	P3 BC0	P5 BC0	P9
05	P2 BC2	P6 BC0	P10
04	P3 BC2	P7 BC0	P11
03	P2 BC1	P4 BC1	P12
02	P3 BC1	P5 BC1	P13
01	P2 BC3	P6 BC1	P14
00	P3 BC3	P7 BC1	P15

Cette structure se complique encore lorsque l'on sait que le premier mot affiché n'est pas nécessairement le premier du bloc. En pratique, un OFFSET de n'importe quelle valeur par pas de 16 bits (pair) peut être ajouté à l'adresse de départ pour indiquer le premier caractère affichable.

Les 10 bits inférieurs de l'adresse mémoire contenue dans les 2 registres R12 et R13 du CRTC 6845 définissent où commence le premier K mots dans la mémoire écran.

De cette dernière information, vous pouvez déduire que :

Par exemple : C7FE, C000 et C002 sont les adresses de début de 3 mots qui peuvent être consécutifs dans le premier bloc.

Afin de clôturer l'étude de la mémoire écran, nous suggérons au lecteur de se servir des connaissances acquises lors de la lecture de ce qui précède pour effectuer les petits essais qui font l'objet de la section suivante.

2.4.3 PROGRAMMATION DIRECTE DE LA MEMOIRE ECRAN

La mémoire écran est très simple à accéder aussi bien en lecture qu'en écriture. Une simple fonction BASIC PEEK vous renseignera sur son contenu et un simple POKE dans une adresse comprise entre C000 et FFFF vous permettra de modifier son contenu.

La mémoire étant sous le contrôle du logiciel interne, pour vous assurer de la position de l'OFFSET, vous devez initialiser votre système avant tout essai.

Exemple 1 : Obtenir un caractère multicolore en mode 1.

Tapez MODE 1

L'écran étant vierge à l'exception du message READY, son coin supérieur gauche correspond à l'adresse C000. A cette adresse et à la suivante, vous trouverez les 2 octets qui constituent la première ligne du R de Ready.

Analysons ces 2 octets.

Faites : PRINT PEEK(&C000),PEEK(&C001)

Vous obtenez 240 et 192.

Décomposez les 2 nombres en binaire pour obtenir :

1 1 1 1 0 0 0 0 1 1 0 0 0 0 0 0

En se servant du tableau de la section 2.4.2, on détermine que P0 , P1 , P2 , P3 , P4 et P5 de BC0 sont à 1, le reste étant à 0 .

Les 6 points de gauche sont donc affichés dans la couleur 1 (BC0 à 1 et BC1 à 0).

Si nous voulons les afficher dans la couleur 2 (BC0 à 0 et BC1 à 1), il suffit d'écrire : 0 0 0 0 1 1 1 1 et 0 0 0 0 1 1 0 0 dans les 2 octets respectifs.

Il suffit donc de taper :

POKE &C000,15 : POKE &C000,12

Exemple 2 : Afficher un point au milieu de l'écran en mode 1

Tapez MODE 1

La ligne du milieu d'écran est la ligne 100.

Pour calculer l'adresse d'une ligne, il suffit d'appliquer la formule suivante :

$$\text{ADRESSE} = \text{C000H} + \text{INT}(\text{NL}/8) * 50\text{H} + (\text{NL} - \text{INT}(\text{NL}/8) * 8) * 800\text{H}$$

NL étant le numéro de la ligne en décimal. Les autres valeurs sont exprimées en hexadécimal. La fonction INT prend le plus petit entier d'un nombre (INTEGER).

Calculons l'adresse de départ de la ligne 100.

$$\text{ADRESSE} = \text{C000H} + \text{INT}(100/8) * 50\text{H} + (100 - \text{INT}(100/8) * 8) * 800\text{H}$$
$$\text{ADRESSE} = \text{C000H} + 12\text{D} * 50\text{H} + 4 * 800\text{H}$$
$$\text{ADRESSE} = \text{C000H} + 3\text{C0H} + 2000\text{H}$$
$$\text{ADRESSE} = \text{E3C0H}$$

La ligne 100 occupe la suite d'adresses comprises entre E3C0 et E3FF. Le milieu se trouve sur les octets E3E8 et E3E9.

Les 2 octets permettant l'affichage de 8 points, le point du milieu est au choix P3 ou P4. Prenons P4, il correspond aux bits B3 et B7 du mot de 16 bits. Allumons le en couleur 1 au moyen de B7 (B7=1 -> 128).

Il suffit donc de faire : POKE &E3E9,128

2.5 Le logiciel interne de gestion d'écran.

2.5.1 INTRODUCTION

Les concepteurs de l'AMSTRAD ont prévu un nombre impressionnant de routines internes qui permettent de simplifier la gestion de l'écran. Toutes ces routines agissent sur le CRTC, sur la VGA ou sur la mémoire écran, mais elles utilisent un grand nombre de mémoires tampons qui permettent de maintenir des sauvegardes partielles des états précédents. Ces routines sont en général situées dans la ROM BIOS (celle du bas de mémoire). Une série de vecteurs situés en mémoire vive permet d'appeler ces routines sans se soucier de la commutation de la ROM.

Les routines peuvent être divisées en quatre grandes classes :

- Les routines écran primaires : elles s'occupent de l'initialisation de l'écran, de la sélection du mode et de la gestion des adresses de la mémoire écran. Ce sont les routines les plus proches de la théorie étudiée au long de ce chapitre.
- Les routines de gestion du mode texte : Elles s'occupent de la gestion du générateur de caractères, du curseur, des encres, des fenêtres, ...
- Les routines de gestion du mode graphique : Elles s'occupent de tracer les points et les lignes, de tester les encres des points, ...
- Les routines en contact direct avec le matériel.

Ces routines vont être décrites de façon succincte, les routines utilisées dans le reste du livre seront décrites en détail lors de leur utilisation. Pour plus de renseignements sur le fonctionnement interne des routines, nous vous invitons à consulter CLEFS POUR L'AMSTRAD de D. MARTIN aux éditions du PSI.

Nous nous contenterons de signaler l'adresse de lancement de la routine ainsi que sa classe générale.

2.5.2 LE GESTIONNAIRE ECRAN PRIMAIRE

A) INITIALISATION

BBFF : Initialisation générale du gestionnaire écran.
BC02 : Remise à leurs valeurs initiales des différents paramètres (encre, vitesse de clignotement...).

B) LIAISON AVEC LE MATERIEL

BC05 : Positionnement de l'OFFSET de début d'écran.
BC08 : Positionnement de la zone mémoire écran (pas de 4000)
BC0B : Lecture de l'OFFSET et de la zone mémoire écran

C) SELECTION DU MODE

BC0E : Positionnement dans un mode précis.
BC11 : Lecture du mode courant.
BC14 : Efface l'écran (CLS).
BC17 : Lecture de la taille écran en caractères.

D) CONVERSION D'ADRESSE

BC1A : Conversion d'une coordonnée physique d'un caractère en position sur l'écran.
BC1D : Conversion d'une coordonnée physique d'un point en position sur l'écran.
BC20 : Calcul de l'adresse écran de l'octet à droite d'un octet donné
BC23 : Idem pour l'octet de gauche.
BC26 : Idem pour l'octet du dessous.
BC29 : Idem pour l'octet du dessus.

E) LES ENCRE

BC2C : Encode une encre pour couvrir tous les pixels d'un octet.
BC2F : Décode une encre en numéro d'encre.
BC32 : Positionne les couleurs d'une encre.
BC35 : Demande les couleurs de l'encre courante.
BC38 : Positionne la couleur du bord.
BC3B : Lecture de la couleur du bord.
BC3E : Positionne la vitesse de clignotement des encres.
BC41 : Lecture de la vitesse de clignotement des encres.

F) DIVERS

BC44 : Colorie un rectangle exprimé en caractère dans une encre déterminée.
BC47 : Colorie un rectangle exprimé en adresse écran dans

une encre déterminée.

BC4A : Inversion de la couleur d'un caractère.

BC4D : SCROLLING haut ou bas avec effacement de la nouvelle ligne.

BC50 : Idem, mais avec recopie de la dernière ligne.

BC53 : Conversion d'une matrice de caractère en un ensemble de points approprié au mode utilisé.

BC56 : Fonction inverse de la précédente.

BC59 : Positionnement de l'écran pour le mode graphique.

BC5C : Ecriture d'un point à l'écran sans se soucier du mode graphique.

BC5F : Dessine une ligne horizontale.

BC62 : Dessine une ligne verticale.

2.5.3 LE GESTIONNAIRE DU MODE CARACTERE

A) INITIALISATION

BB4E : Initialisation mode texte.

BB51 : Remise des paramètres à leurs valeurs initiales.

BB54 : Autorise l'écriture de caractères.

BB57 : Interdit l'écriture de caractères.

B) CARACTERES

BB5A : Sortie d'un caractère ou d'un code de contrôle.

BB5D : Ecriture d'un caractère sur l'écran.

BB60 : Lecture du caractère courant écran sous le curseur.

BB63 : Sélectionne ou interdit l'utilisation du gestionnaire graphique pour l'écriture des caractères.

C) FENETRES

BB66 : Positionne la taille de la fenêtre courante.

BB69 : Lecture de la taille de la fenêtre courante.

BB6C : Efface le contenu de la fenêtre courante.

D) CURSEUR

BB6F : Positionne le curseur en horizontal.

BB72 : Positionne le curseur en vertical.

BB75 : Positionne le curseur en horizontal et en vertical.

BB78 : Lecture de la position du curseur et du nombre de SCROLLING effectué par la fenêtre courante.

BB7B : Autorise l'affichage du curseur (pour les programmes utilisateurs).

BB7E : Interdit l'affichage du curseur (pour les programmes utilisateurs).

BB81 : Autorise l'affichage du curseur (pour la ROM).

BE84 : Interdit l'affichage du curseur (pour la ROM).
BE87 : Teste si une position curseur est dans la fenêtre.
BE8A : Place un pavé curseur à la position du curseur.
BE8D : Enlève le pavé curseur à la position du curseur.

E) ENCRE

BE90 : Positionne l'encre crayon courante.
BE93 : Lecture de l'encre crayon courante.
BE96 : Positionne l'encre papier courante.
BE99 : Lecture de l'encre papier courante.
BE9C : Inverse l'encre du crayon et l'encre du papier.
BE9F : Autorise ou interdit l'écriture sur le fond.
BBA2 : Lecture de l'état du fond pour l'écriture (autorisé ou interdit).

F) MATRICES DE CARACTERES

BBA5 : Calcul de l'adresse de la matrice d'un caractère et détermination de son type (définie par l'utilisateur ou en ROM).
BBA8 : Construit une matrice pour un caractère (utilisé pour les caractères définis par l'utilisateur).
BBAB : Positionne l'adresse de la table des caractères définis par l'utilisateur.
BBAE : Lecture de l'adresse de la table des caractères définis par l'utilisateur.

G) DIVERS

BBB1 : Lecture de l'adresse de la table des codes de contrôle.
BBB4 : Sélectionne un flux vidéo.
BBB7 : Echange les descripteurs de deux flux vidéos.

2.5.4 LE GESTIONNAIRE DU MODE GRAPHIQUE.

A) INITIALISATION

BBBA : Initialisation du gestionnaire graphique.
BBBD : Remise de la configuration standard.

B) POSITIONNEMENT

BBC0 : Déplacement vers une position absolue.
BBC3 : Déplacement vers une position relative.
BBC6 : Lecture de la position courante.
BBC9 : Positionnement de l'origine des coordonnées.
BBCC : Lecture de l'origine des coordonnées.

C) FENETRES

BBCE : Positionne les bords gauche et droit d'une fenêtre.
BBD2 : Positionne les bords haut et bas d'une fenêtre.
BBDE : Lecture de la position des bords gauche et droit.
BBDB : Lecture de la position des bords haut et bas.
BBDB : Nettoie la fenêtre graphique courante.

D) ENCREES

BBDE : Positionne la couleur d'un crayon graphique.
BBE1 : Lecture de la couleur du crayon.
BBE4 : Positionne la couleur du papier.
BBE7 : Lecture de la couleur du papier.

E) DESSIN ET TESTS

BBEA : Dessine un point à une position absolue.
BBED : Dessine un point à une position relative.
BBF0 : Teste la couleur d'un point à une position absolue.
BBF3 : Teste la couleur d'un point à une position relative.
BBF6 : Trace une ligne à une position absolue.
BBF9 : Trace une ligne à une position relative.
BBFC : Ecrit un caractère sur l'écran à la coordonnée graphique courante.

2.5.5 LES ROUTINES EN CONTACT AVEC LE MATERIEL

BD19 : Attente de génération du signal de retour du spot.
BD1C : Positionnement d'un mode dans la VGA.
BD1F : Positionnement de l'OFFSET de la mémoire écran.
BD22 : Positionne toutes les encres dans la même couleur.
BD25 : Positionne la couleur de toutes les encres et du bord.

Ce chapitre est terminé, mais nous reviendrons sur l'utilisation des routines internes de gestion de l'écran dans le chapitre réservé aux RSXs ainsi que dans les programmes, trucs et astuces.

Abandonnons quelques instants le plaisir des yeux par l'image et le graphique pour nous consacrer à celui des oreilles. Autrement dit, passons à l'étude du gestionnaire sonore.

LE GÉNÉRATEUR SONORE AY3-8912

3.1 Généralités

Le générateur sonore AY3-8912 de Général Instrument, en abrégé PSG (Programmable Sound Générateur), est un circuit LSI (circuit à haute intégration) qui peut produire une grande variété de sons complexes sous le contrôle d'un programme approprié.

Le FSG est un composant relativement facile à interfacer avec n'importe quel système à microprocesseur. Sa flexibilité est telle qu'il est souvent utilisé dans toutes sortes d'applications les plus diverses.

Comme les synthétiseurs musicaux, les alarmes et les signalisations sonores ou les modems utilisent la technique FSK (Frequency Shift Keying).

La sortie sonore analogique (c'est à dire un signal non digital) est effectuée par l'intermédiaire d'un convertisseur DAC (Digital Analogic Converter) sur quatre bits, celui-ci permet une grande variété d'effets sonores.

Une des caractéristiques principales du circuit est qu'une fois ses commandes de génération reçues, il gère lui-même les effets sonores laissant le processeur libre pour continuer d'autres tâches. Ainsi, le PSG peut produire des sons relativement longs en n'affectant en rien la vitesse d'exécution du programme.

Le PSG possède trois voies mixables et permet donc la sortie de trois sons simultanés, donc la création d'un accord musical simple majeur ou mineur.

Le PSG est un coprocesseur dont la gestion se fait au moyen de différents registres programmables. Ces registres sont au nombre de 16 et seront décrits en détail au cours de ce chapitre.

3.2 La théorie du son.

Pour bien comprendre la programmation et l'utilisation du générateur sonore, il est indispensable d'analyser dans les grandes lignes tous les paramètres qui définissent un phénomène sonore ou, plus simplement, une note de musique.

3.2.1 QUALITES DU SON.

Un son consiste en la propagation, à partir d'une source (en ce qui nous concerne, la membrane du haut parleur), d'ondes matérielles périodiques longitudinales avec une vitesse dont la grandeur dépend du milieu de propagation.

Exemple: Vibration d'un ressort après avoir exercé sur lui une compression ou une traction.

On entend par qualité du son, les diverses caractéristiques par lesquelles différents sons se distinguent les uns des autres.

Elles sont au nombre de trois: la HAUTEUR, le VOLUME, et le TIMBRE.

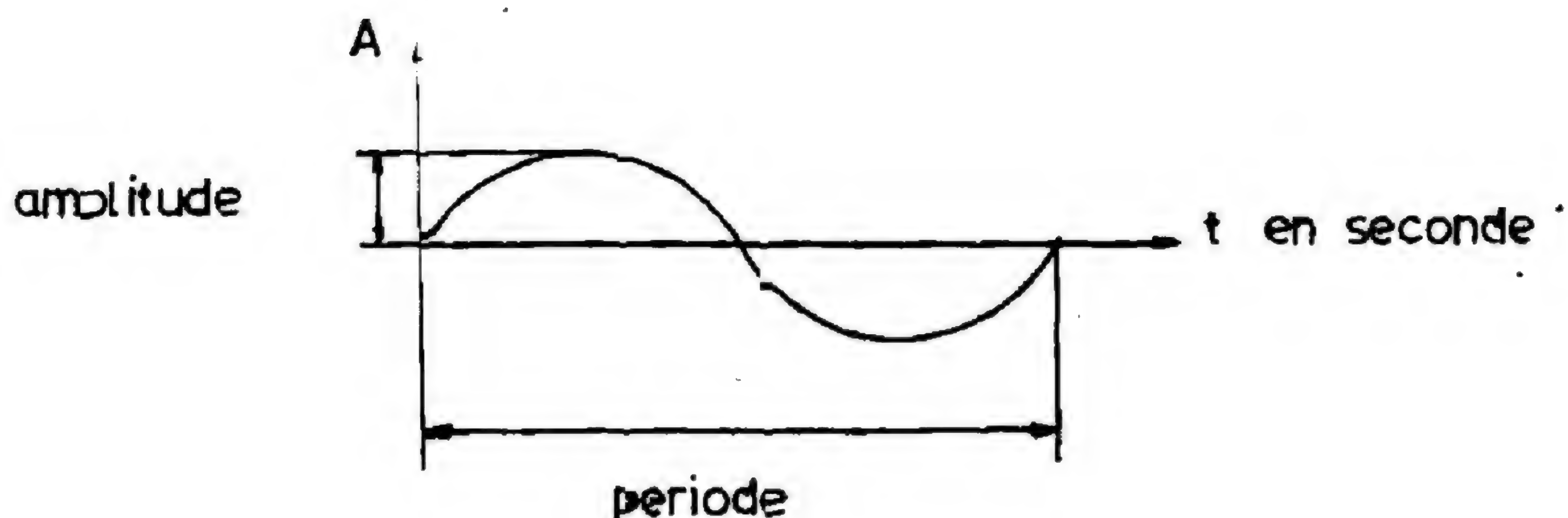
3.2.2 LA HAUTEUR.

Lorsque nous frappons successivement quelques touches d'un piano, nous percevons des sons différents les uns des autres. Les sons se distinguent par leur hauteur ou leur fréquence.

Les sons graves correspondent aux basses fréquences et les sons aigus aux fréquences élevées.

Dans une note de musique, la hauteur est le facteur essentiel.

Une note musicale peut être décrite comme une oscillation qui se caractérise par une fréquence, une période et une amplitude.



La période est le temps, exprimé en secondes, d'une oscillation complète.

L'amplitude dépend du volume et n'entre pas dans la définition de la hauteur d'une note.

La fréquence et la période sont liées entre elles par le rapport suivant:

$$\text{FREQUENCE} = 1 / \text{PERIODE}$$

où la fréquence est exprimée en Hertz et la période, en secondes.

Notons que l'impression musicale résultant de l'audition de deux sons de hauteurs différentes dépend non pas des valeurs absolues de celles-ci mais bien de leur rapport. C'est ce rapport que l'on appelle INTERVALLE DE DEUX TONS.

Lorsque l'un des sons présente une fréquence double de celle de l'autre, l'intervalle est appelé OCTAVE.

La gamme dite naturelle est formée de différents sons présentant les intervalles suivants par rapport au premier son:

DO	RE	MI	FA	SOL	LA	SI	DO
1	9/8	5/4	4/3	3/2	5/3	15/8	2

Les octaves se suivent en présentant les mêmes intervalles. Les différents octaves sont affectés respectivement des indices 1,2,3,... ou des indices -1,-2,-3,... suivant qu'ils sont supérieurs ou inférieurs à l'octave 0, octave de référence.

3.2.3 LE VOLUME.

Une même touche de piano produira un effet sonore différent suivant qu'elle est frappée légèrement ou fortement.

Dans le premier cas, le son perçu est faible; dans le second, il est fort. Les deux sons diffèrent par leur VOLUME, c'est à dire, leur intensité.

Le Volume détermine donc la grandeur de l'effet sonore.

3.2.4 LE TIMBRE.

L'oreille humaine distingue des sons de même hauteur et de même intensité mais émis par des instruments différents. Le LA du violon est perçu différemment du LA du piano.

Cette différence est due au fait qu'une note émise par un instrument n'est jamais pure (ou simple) mais est composée d'une multitude de sons simples, appelés HARMONIQUES, superposés au son pur initial, appelé FONDAMENTAL.

Le timbre d'un son dépend des fréquences et des intensités relatives des harmoniques qui accompagnent le son fondamental.

3.2.5 LES BRUITS.

On distingue les bruits des sons musicaux par le fait que les premiers ne sont pas périodiques, c'est à dire, qu'ils ne se reproduisent pas exactement à intervalle régulier dans le temps.

La distinction entre une note et un bruit n'est pas toujours très nette. Bien que l'oreille humaine soit sensible à tous les sons dont la fréquence est comprise entre 20 et 20.000 hertz, l'effet musical n'est plus perçu pour les sons périodiques dont la fréquence est trop basse ou trop élevée. De plus, les limites varient d'un auditeur à l'autre.

3.2.6 DUREE ET ATTAQUE DE LA NOTE.

La durée d'un son présente une importance non négligeable quant à l'interprétation que l'oreille humaine pourra en faire.

Pour l'oreille, la durée d'un son dépendra de l'état physiologique de l'individu et des durées relatives des sons qui ont précédé le son considéré.

De plus, il faut noter que la perception sonore dépend également de l'attaque ou de la chute d'un son.

On entend par attaque (chute) la vitesse à laquelle le son considéré atteint un volume déterminé.

3.2.7 PARAMETRES DEFINISSANT UN SON DANS LE CPC.

En Basic Amstrad, la hauteur d'un son est définie par la période de Ton.

La fréquence (en Hertz) = $62.500 / PT$ (période de Ton, en secondes)

Exemple: pour obtenir une fréquence de 1000 hertz, la période de ton doit être égale à 62,5.

$$F(\text{hz}) = 62.500 / PT = 1000 \text{ hz.}$$

La variation de volume est confiée à l'instruction ENV (enveloppe de volume). Les enveloppes de volume déterminent l'attaque, la durée et la chute du son. L'Amstrad Basic permet de définir jusqu'à 15 enveloppes.

La durée définie dans la commande SOUND est exprimée en centièmes de seconde. Elle peut être aussi déterminée par la durée de l'enveloppe de volume.

La commande basic ENT définit les enveloppes de ton. Sa structure est la même que celle de ENV. Elle permet d'obtenir des variations de hauteur du son émis.

Nous terminerons ce paragraphe par la formule qui donne la fréquence des différentes notes dans les différentes octaves à partir du LA international de 440 hertz.

$$\text{Fréquence (Hz)} = 440 * 2^{\text{exp}(\text{NUM.OCT} + (\text{N}-10)/12)}$$

où NUM.OCT est le numéro de l'octave
N est le numéro de la note. Les notes sont
numérotées de 1 à 10. Le LA est la 10^e note.

Exemple: Nous désirons obtenir le SOL de l'octave n° 2.
SOL est la 8^e note.

$$F \text{ (Hz)} = 440 * 2^{\exp (2+(8-10)/12)} \\ = 1567,982 \text{ hertz.}$$

Remarque: la formule figurant dans le manuel est
erronée:

$$F = 440 * (2^{\exp \text{ Octave} + (10-N)/12})$$

De plus, la table donnant les différentes périodes est
décalée d'une octave vers le bas. L'octave 0 est en réalité
l'octave 1. Le LA de référence correspond à PT = 142.

Après ce rappel un peu long mais nécessaire sur les
notions fondamentales de la théorie du son, nous pouvons
enfin aborder l'étude et la programmation du générateur
sonore AY3-8912.

3.3 Structure interne du PSG

Le PSG est composé des éléments suivants:

a) **GENERATEURS SONORES:** au nombre de trois, ils sont respectivement appelés canal A, canal B et canal C. Ils produisent un signal carré dont la fréquence est programmable.

b) **LE GENERATEUR DE BRUIT:** Il produit une modulation de fréquence aléatoire.

c) **LE MELANGEUR (MIXER):** Il permet de combiner les sorties des trois canaux A, B et C du générateur sonore avec le générateur de bruit.

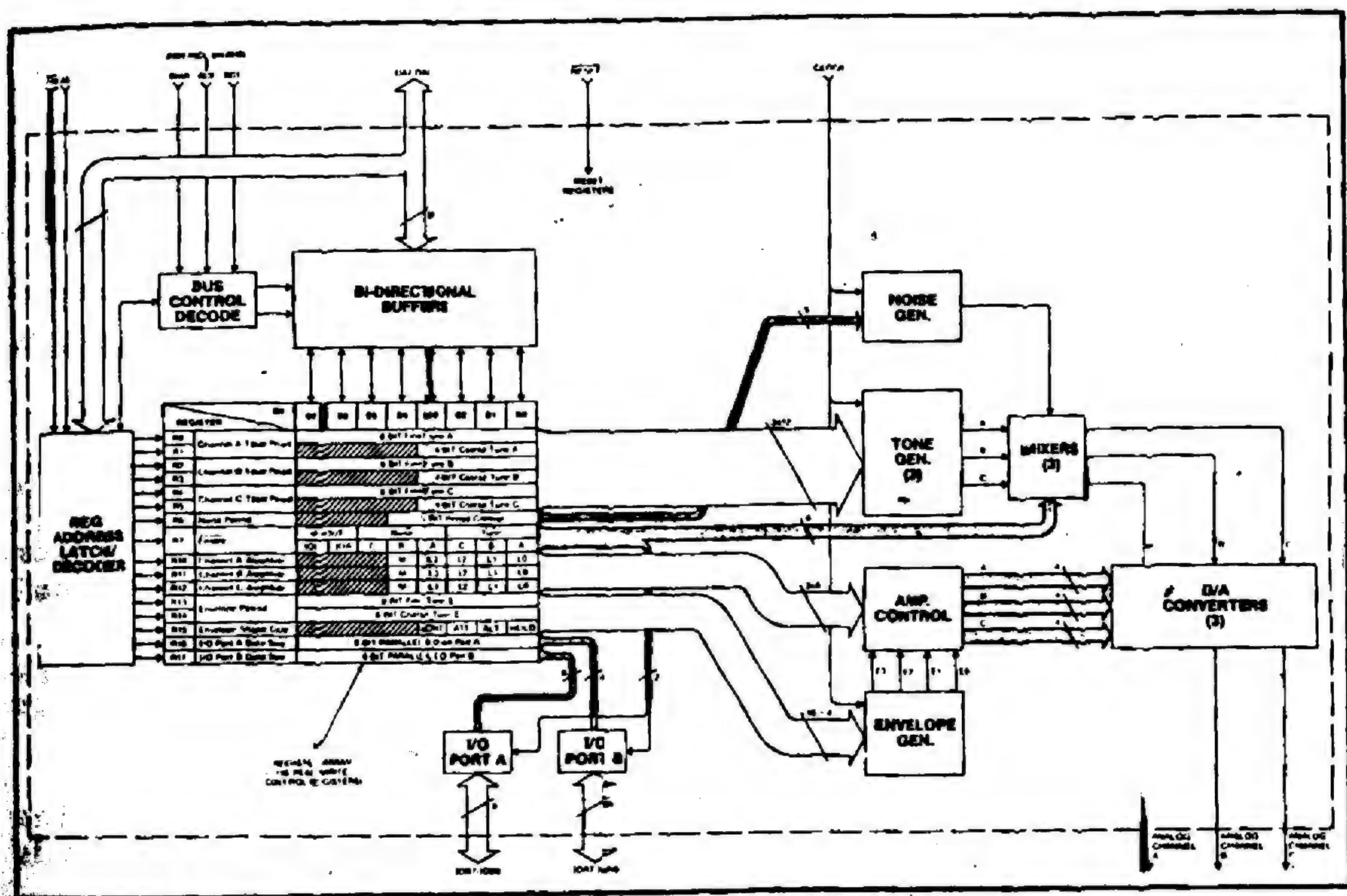
d) **LE CONTROLEUR D'AMPLITUDE:** Il fournit au D/A (Digital Analog convert) la possibilité de contrôler l'amplitude par un modèle fixe ou variable. Le modèle d'amplitude fixe est contrôlé par le microprocesseur lui-même tandis que le modèle d'amplitude variable est obtenu par l'utilisation du générateur d'enveloppe.

e) **LE GENERATEUR D'ENVELOPPE:** Il produit un modèle de variation d'amplitude appelé enveloppe qui peut être utilisé pour moduler la sortie de chaque mixer.

f) **LES CONVERTISSEURS DIGITAUX/ANALOGIQUES: D/A:** Les trois convertisseurs D/A produisent un signal de sortie sur 16 niveaux déterminés par le contrôleur d'amplitude.

g) **LES PORTS D'ENTREE/SORTIE:** Le générateur sonore AY3-8912 possède un port d'entrée/sortie. Ce port ne joue aucun rôle dans la production sonore. Il sera analysé en même temps que le PPI (Programmable Périphéral Interface) au cours du chapitre 5.

Schéma représentant la structure interne du PSG AY3-8912:



3.1 LES DIFFERENTS REGISTRES DU PSG.

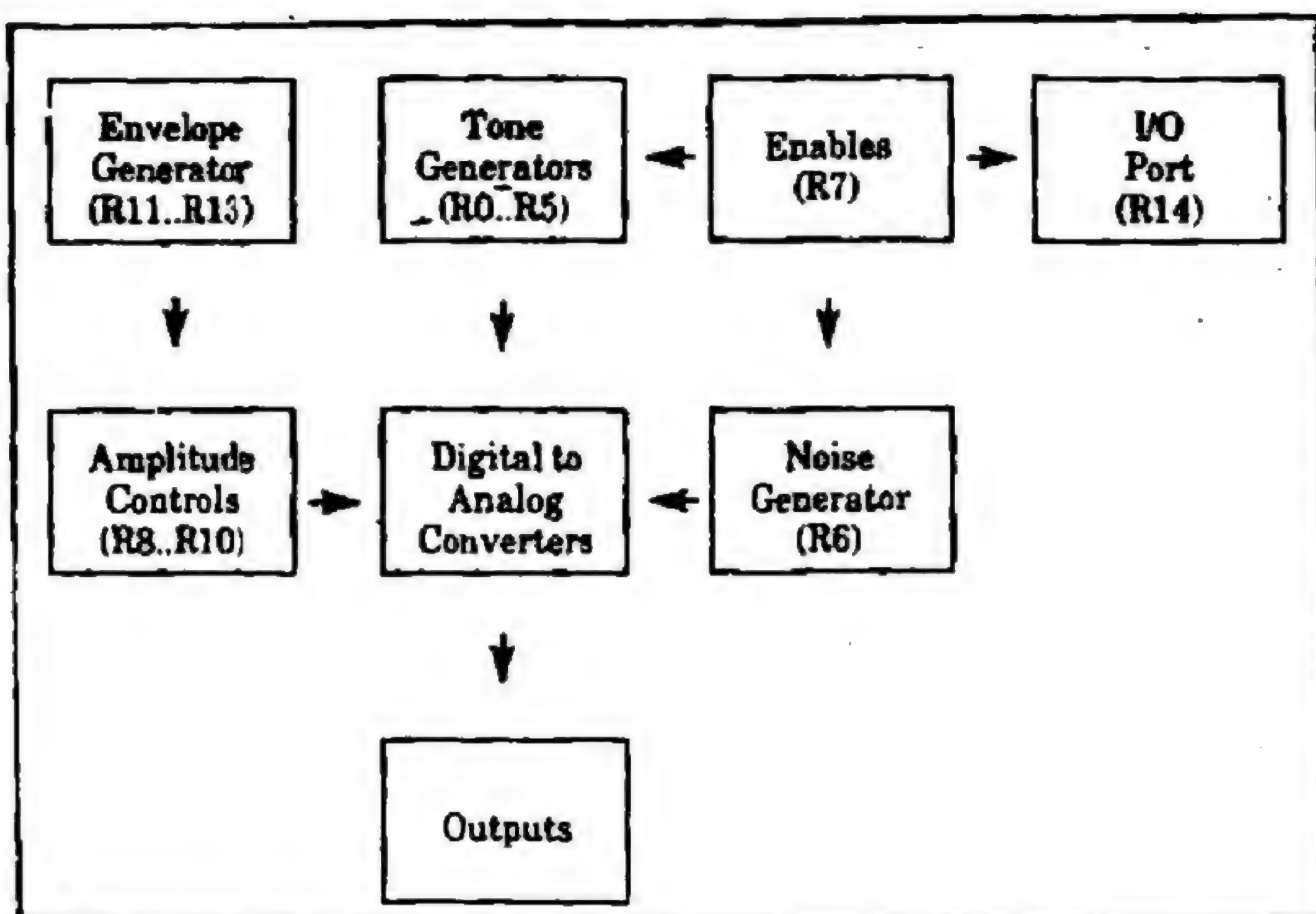
Les registres qui permettent la programmation du PSG sont au nombre de 16, numérotés de R0 à R15.

Les registres R14 et R15 servent à la gestion des ports d'entrée/sortie et seront analysés par la suite.

Remarque: Pour AY3-8912, il n'y a qu'un seul port de sortie. Le registre R15 n'est pas utilisé.

Pour produire un son, une combinaison des registres R0 à R15 doit être introduite dans le PSG. Chaque son doit être analysé de façon à séparer les différents paramètres qui le définissent. C'est à dire: la composante de bruit, de son, la fréquence, la forme et la durée des enveloppes.

Une fois cette analyse terminée, les registres peuvent être chargés et le son produit. Le schéma bloc figurant ci-après montre les interactions entre les différentes sections du PSG.



3.3 2 LES REGISTRES R0 à R5.

Les registres R0 à R5 définissent la fréquence du son émis. Ils sont divisés en trois paires: R0-R1 pour le canal A, R2-R3 pour le canal B et R4-R5 pour le canal C.

Les registres R1, R2 et R4 sont les registres de réglage fin de la fréquence et les 8 bits sont utilisés.

Les registres R1, R3 et R5 quant à eux sont les registres de réglage grossier (seuls les 4 bits de poids faible LSB sont utilisés).

Les valeurs chargées dans les registres R0, R2 et R4 sont donc comprises entre 0 et 15 (00 et FF en hexadécimal).

Dans le PSG, la fréquence d'un son est déterminée en divisant premièrement la fréquence de l'horloge interne par 16 et puis par F, qui est la fréquence à programmer.

On applique la formule suivante:

$$PT = 1\ 000\ 000 / (16 * F) \text{ où } PT \text{ est la période de Ton.}$$

La valeur résultante des paires de registres étant codée sur 12 bits, PT doit être arrondi à l'unité avant d'être exprimé sur 12 bits au moyen de la fonction

$$PT = \text{BIN\$}(F, 12) = \begin{array}{ccc} \text{xxxx} & \text{xxxx} & \text{xxxx} \\ \backslash\text{---}/ & \backslash\text{-----}/ & \\ \text{RH} & & \text{RL} \end{array}$$

Les huit bits de droite sont transmis dans les registres R0, R2 ou R4 et les quatre bits de gauche, dans les registres R1, R3 ou R5.

Ajustement grossier -canal- ajustement fin

B7 B6 B5 B4 B3 B2 B1 B0

B7 B6 B5 B4 B3 B2 B1 B0

Non utilisé

PT11 PT10 PT9 PT8 PT7 PT6 PT5 PT4 PT3 PT2 PT1 PT0

12 bits pour le générateur sonore.

Une autre façon de procéder pour charger les paires de registres consiste à calculer RL et RH comme suit:

$$RL = PT - (INT(PT/256)*256)$$

et $RH = INT(PT/256)$ ou encore: $RH = PT \setminus 256$
(c'est bien le signe
et non /)

Il suffit alors de transmettre RL dans R0, R2 ou R4 et RH dans R1, R3 ou R5.

Exemple: Soit $F = 440$ Hz (Le LA international).

	$PT = 1\,000\,000 / (16*440)$
	$PT = 1\,000\,000 / 7\,040$
	$PT = 142,046$
On arrondit	$PT = 142$
On calcule	$RL = 142 - (INT(142/256)*256)$
	$RL = 142 - (INT(0,555 * 256)$
	$RL = 142 - (0 * 256)$
	$RL = 142$
	$RH = INT(142/256)$
	$RH = 0$

Si c'est le canal A qui doit être programmé, R0 sera égal à 142 et R1, à 0.

Détermination de F minimum et de F maximum

Comme PT (période de ton) est exprimé sur 12 bits, la valeur que l'on peut charger ne peut excéder 4095, c'est à dire $(2 \exp 12 - 1)$ et la valeur minimale est égale à 1.

1 donne F max et 4095 donne F min.

$$1 = 1000000 / (16 * F_{\max}) ; F_{\max} = 1000000 / (16 * 1) = 62.500 \text{ hz}$$

$$4095 = 1000000 / (16 * F_{\min}) ; F_{\min} = 1000000 / (16 * 4095) = 15,26 \text{ hz}$$

Il est évident qu'une fréquence F_{max} de l'ordre de 62.500 hz est, comme nous l'avons vu dans le paragraphe traitant de la théorie du son, imperceptible par l'oreille humaine.

De plus, la bande passante des petits amplis audio ou de télévision dépasse rarement les 5000 hz. Dès lors, nous nous fixerons comme valeur maximum de fréquence cette valeur (5000 hz).

$$PT_{max} = 1\ 000\ 000 / (16 * 5000) = 12,5$$

Les valeurs de PT seront comprises entre 12 et 4095.

3.3.3 LE REGISTRE R6.

Le registre R6 du PSG permet de programmer une fréquence de bruit de la façon suivante:

Seuls les 5 bits de poids faible du registre R6 sont utilisés pour programmer le générateur de bruit.

Registre de période de bruit

B7 B6 B5 B4 B3 B2 B1 B0

non utilisé 5 bits pour le générateur de bruit

La formule appliquée pour trouver la période de bruit est semblable à celle utilisée pour calculer la période de ton (PT).

Soit PB pour période de bruit:

$PB = 1\ 000\ 000 / (16 * F_b)$ où F_b est la fréquence de bruit désirée.

Comme seuls les cinq premiers bits de R6 sont utilisés pour déterminer PB, les valeurs de PB_{max} et PB_{min} seront comprises entre 1 et $(2^{exp\ 5} - 1)$, soit 31.

Par la formule, on détermine F_{bmax} et F_{bmin} :

$$F_{bmax} = 1\ 000\ 000 / (16 * PB_{min})$$

$$F_{bmax} = 62\ 500\ hz$$

$$F_{bmin} = 1\ 000\ 000 / (16 * PB_{max})$$

$$F_{bmin} = 2\ 016\ hz$$

Il est à noter que les restrictions applicables pour F_{max} sont aussi à observer pour F_{bmax} (fréquence de bruit maximum).

3.3.4 LE REGISTRE R7.

Le registre R7 permet de contrôler la fonction de mixage entre les trois générateurs sonores et les trois générateurs de bruits. Le mixage, comme décrit au paragraphe 3.3, autorise la combinaison de chaque canal A, B ou C avec un générateur de bruits. Le registre R7 contrôle également les ports d'entrée/sortie A et B dont nous parlerons par la suite.

Remarque: Nous rappelons que dans l'AY3-8912, seul le port A existe.

Registre de contrôle du mixage

	B7 B6		B5 B4 B3			B2 B1 B0		
I/O PORT	B	A	C	B	A	C	B	A
Fonction	INPUT ENABLE		NOISE ENABLE			TONE ENABLE		

Remarques:

1) Mettre un canal sur OFF ne suffit pas pour arrêter l'émission de celui-ci. Il faut écrire 0 dans le registre de contrôle d'amplitude (voir ci-dessous).

2) Attention, les bits du registre R7 sont actifs bas.

Exemple: je désire produire sur le canal A du son et pas de bruit, sur le canal B, du bruit et pas de son et sur le canal C, du bruit et du son.

Je dois donc charger le registre R7 avec la valeur suivante:

Val. déc.	128	64	32	16	8	4	2	1
bit	B7	B6	B5	B4	B3	B2	B1	B0
	X	X	0	0	1	0	1	0

(XX) = sans importance.

R7 = 10, en base 10 ou R7 = 0AH, en base 16

Tableau résumant les effets du registre R7:

BIT = 0		BIT = 1	
B0	SON sur le canal A (ON)	SON sur le canal A (OFF)	
B1	SON sur le canal B (ON)	SON sur le canal B (OFF)	
B2	SON sur le canal C (ON)	SON sur le canal C (OFF)	
B3	BRUIT sur le canal A (ON)	BRUIT sur le canal A (OFF)	
B4	BRUIT sur le canal B (ON)	BRUIT sur le canal B (OFF)	
B5	BRUIT sur le canal C (ON)	BRUIT sur le canal C (OFF)	
B6	PORT A en entrée	PORT A en sortie	
B7	PORT B n'existe pas	PORT B n'existe pas.	

3.3.5 LES REGISTRES R8 à R10.

L'amplitude du signal sonore émis par les trois convertisseurs D/A (un pour chacun des canaux A, B et C) est déterminée par le contenu du registre R8 pour le canal A, du registre R9 pour le canal B et du registre R10 pour le canal C.

Seuls les quatre bits les moins significatifs (B3-B0) de chacun des registres sont utilisés. Ils permettent donc des valeurs comprises entre 0 et 15. La valeur 0 chargée dans l'un des registres correspond à un volume (nul). L'amplitude maximale est obtenue en chargeant le registre avec la valeur 15. Le cinquième bit (B4) est utilisé pour sélectionner le mode de fonctionnement du contrôle d'amplitude.

Si le bit B4 est égal à 0, l'amplitude ne varie pas. Si B4 est égal à 1, la variation d'amplitude est confiée au générateur d'enveloppe. (voir ci-dessous).

Registre de contrôle d'amplitude:

B7 B6 B5	B4	B3 B2 B1 B0
Non utilisé	mode	niveau d'amplitude (4 bits)

3.3.6 LES REGISTRES R11 ET R12

Les registres R11 et R12 permettent au PSG de contrôler la période de l'enveloppe. Un calcul similaire à celui des registres R0 à R5 fournit les valeurs à charger dans les registres R11 et R12.

Soit PE pour la période d'enveloppe et Fe pour la fréquence enveloppe:

$$PE = 1\ 000\ 000 / (256 * Fe)$$

Les 8 bits des registres R11 et R12 sont utilisés. La valeur résultante est donc codée sur 16 bits et peut varier de 0 à 65 535 ($2^{16} - 1$).

Le registre R11 du PSG définit l'ajustement fin de la période d'enveloppe tandis que R12 permet un ajustement grossier.

ajustement grossier R12

B7 B6 B5 B4 B3 B2 B1 B0

PE15 à PE8

ajustement fin R11

B7 B6 B5 B4 B3 B2 B1 B0

PE7 à PE0

PE (16 bits pour le générateur d'enveloppe)

Par la formule, on peut calculer Pe max et Pe min des enveloppes possibles:

$$Fe = 1\ 000\ 000 / (256 * PE)$$

$$Fe = 1/Pe = 1\ 000\ 000 / (256 * PE)$$

$$Pe = (256 * PE) / 1\ 000\ 000$$

Pe max = $(256 * 65535) / 1\ 000\ 000 = 16,777$ secondes, ce qui correspond à une fréquence d'enveloppe Fe=0,0596 Hz.

Pe min = $(256 * 1) / 1\ 000\ 000 = 0,000\ 256$ secondes, ce qui correspond à une fréquence d'enveloppe Fe=3906,25 Hz.

3.3.7 LE REGISTRE R13.

Le registre R13 contrôle les différentes formes de modulation utilisées par le PSG. Si le bit (B4) décrit dans les registres R8 à R10 est à 1, la modulation a lieu, sinon, la programmation du registre 13 est ignorée. Seuls les 4 bits les moins significatifs sont utilisés. Ils déterminent chacun un paramètre de modulation.

R13 : Contrôle de forme de modulation.

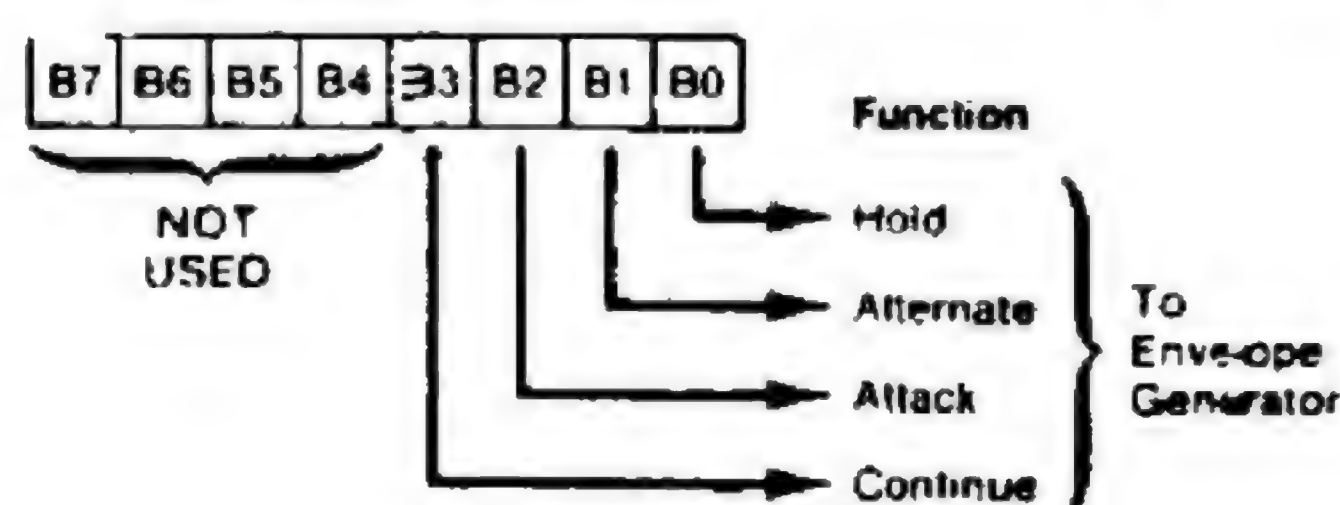


Table des modulations.

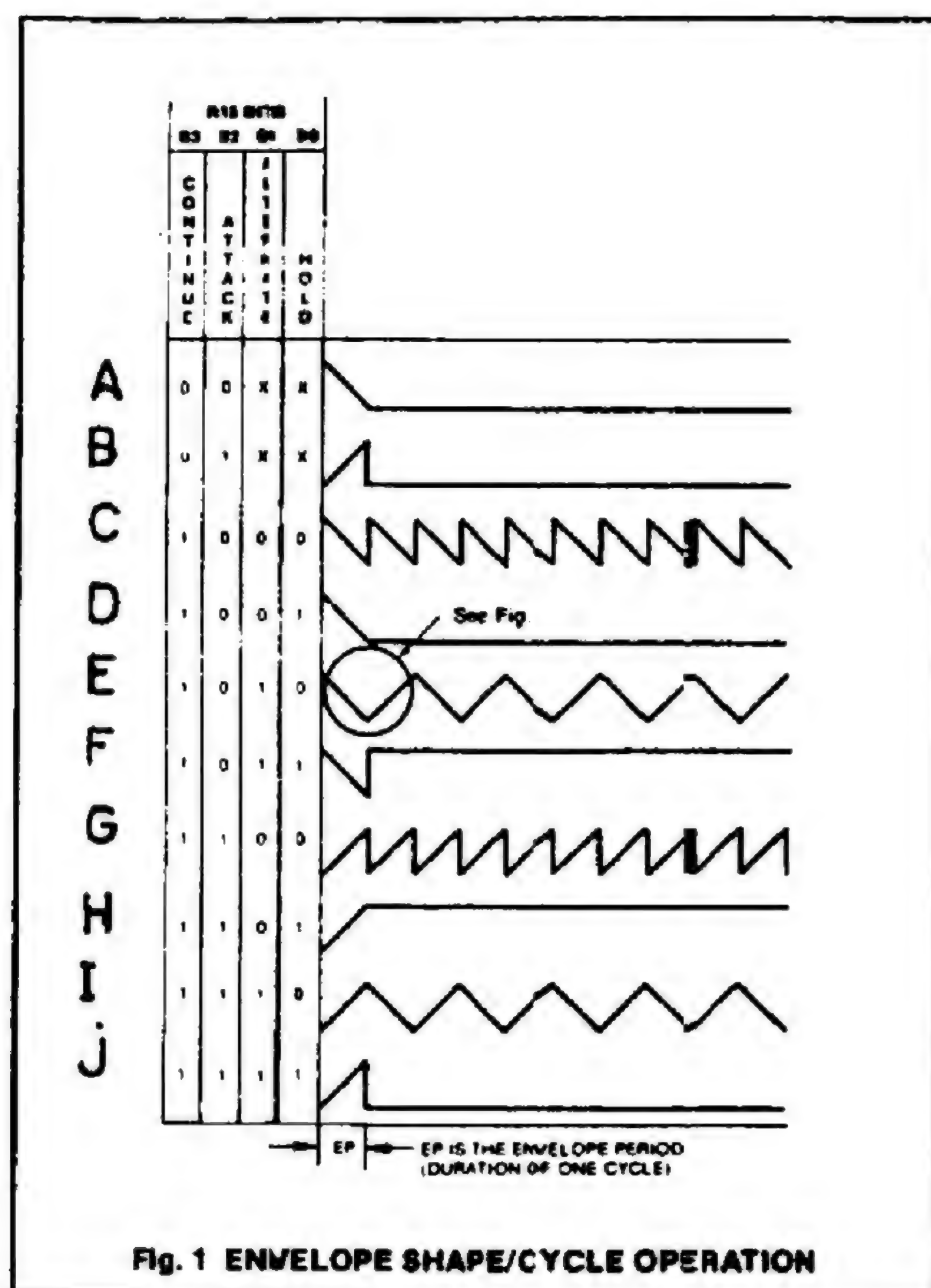
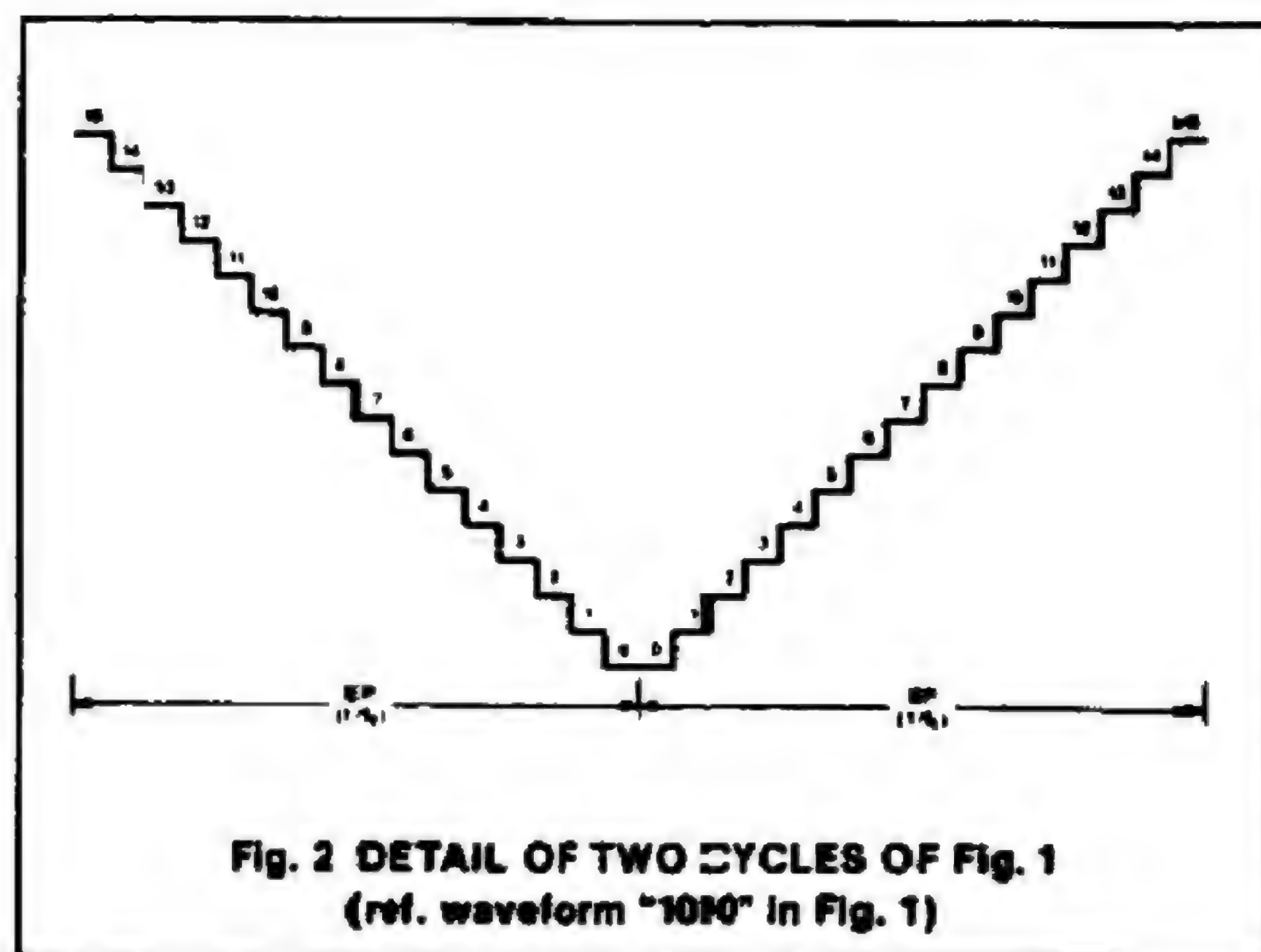
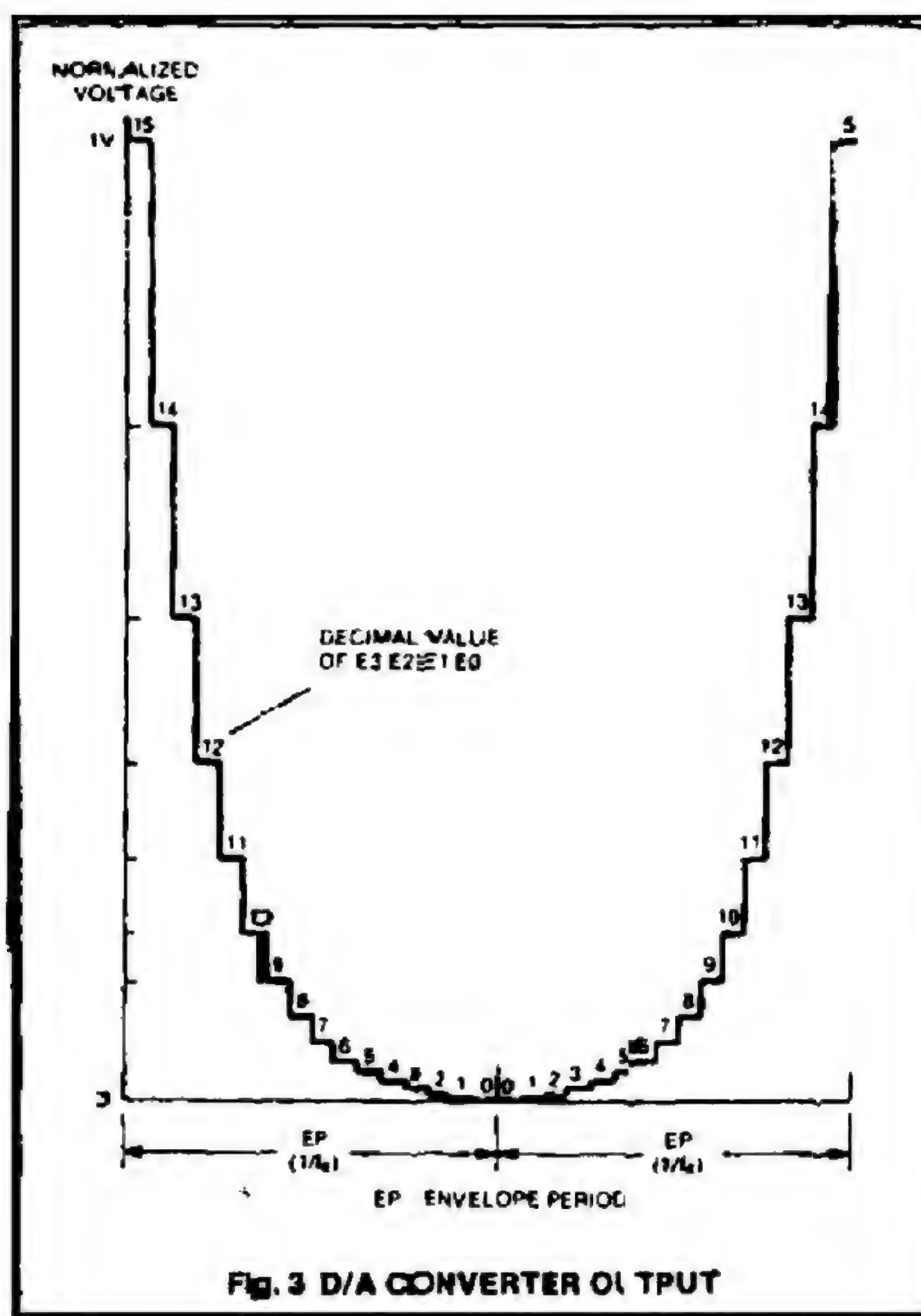


Fig. 1 ENVELOPE SHAPE/CYCLE OPERATION





3.3.8 LES REGISTRES R14 et R15.

Nous avons vu, lors de l'étude du registre R7, que les bits B6 et B7 définissent le sens de la transmission des données (0=entrée, 1=sortie) sur les ports d'entrée/sortie.

Le registre R14, est utilisé pour l'écriture et la lecture du PORT A. Le PORT B n'existant pas dans l'AY3-8912, le registre R15 n'est pas utilisé.

Dans l'Amstrad, le Port A du PSG est utilisé pour la gestion de l'entrée clavier. Nous étudierons plus en détails la gestion clavier dans le chapitre 4 traitant du PPI (Programmable Peripheral Interface).

3.4 Programmation des registres R0 à R13.

La programmation du générateur sonore peut se faire en BASIC de trois manières différentes. la première consiste à utiliser l'instruction SOUND, la seconde, à faire appel à une routine en langage machine se trouvant dans la ROM et la troisième, à utiliser les instructions OUT et INP.

Cette dernière méthode est un peu plus compliquée. Elle demande la connaissance du fonctionnement du PPI et de sa programmation. Elle sera développée au cours du chapitre 4.

A) Rappel des commandes SOUND, ENV et ENT.

1° SOUND A,B,C,D,E,F,G

Où A = statut des canaux:

Décimal	BIT	Commande
1	0 lsb	son dirigé sur le canal A
2	1	son dirigé sur le canal B
4	2	son dirigé sur le canal C
8	3	rendez-vous avec le canal A
16	4	rendez-vous avec le canal B
32	5	rendez-vous avec le canal C
64	6	maintien
128	7 msb	"Flush"

B = Période de ton: valeur de 0 à 4095

C = durée: valeur de -32 768 à +32 767

- Pour les valeurs > 0, la durée est exprimée en centièmes de seconde (0,01 sec).

- Pour une valeur = 0, la durée est déterminée par l'enveloppe.

- Pour les valeurs < 0, la valeur absolue donne le nombre de répétitions de l'enveloppe de volume.

D = Volume: valeur de 0 à 15 ou de 0 à 7 s'il n'y a pas d'enveloppe.

E = enveloppe de volume: valeur de 0 à 15.

F = enveloppe de ton: valeur de 0 à 15.

ENV H, I, J, K, I1, J1, K1, ..., I5, J5, K5

où H = numéro d'enveloppe: valeur de 1 à 15

I = nombre de pas: valeur de 0 à 127

J = taille de pas: valeur de -128 à +127.

K = durée du pas: valeur de 0 à 255.

3. ENT L, N, M, O

Même structure que ENV, mais donne une enveloppe de variation de fréquence, c'est à dire de hauteur.

21 Pour de plus amples informations, nous demandons au lecteur de se référer au manuel BASIC.

EXAMPLE N°1: 10 ENV 1,15,1,40,15,-1,40
20 SOUND 1,142,30(0,*,1

142 correspond au LA de l'octave 0. La durée totale du SOUND est de $3000 \times 0,01$. C'est à dire 30 secondes. La durée d'un pas est de $40 \times 0,01$. C'est à dire 0,4 secondes.

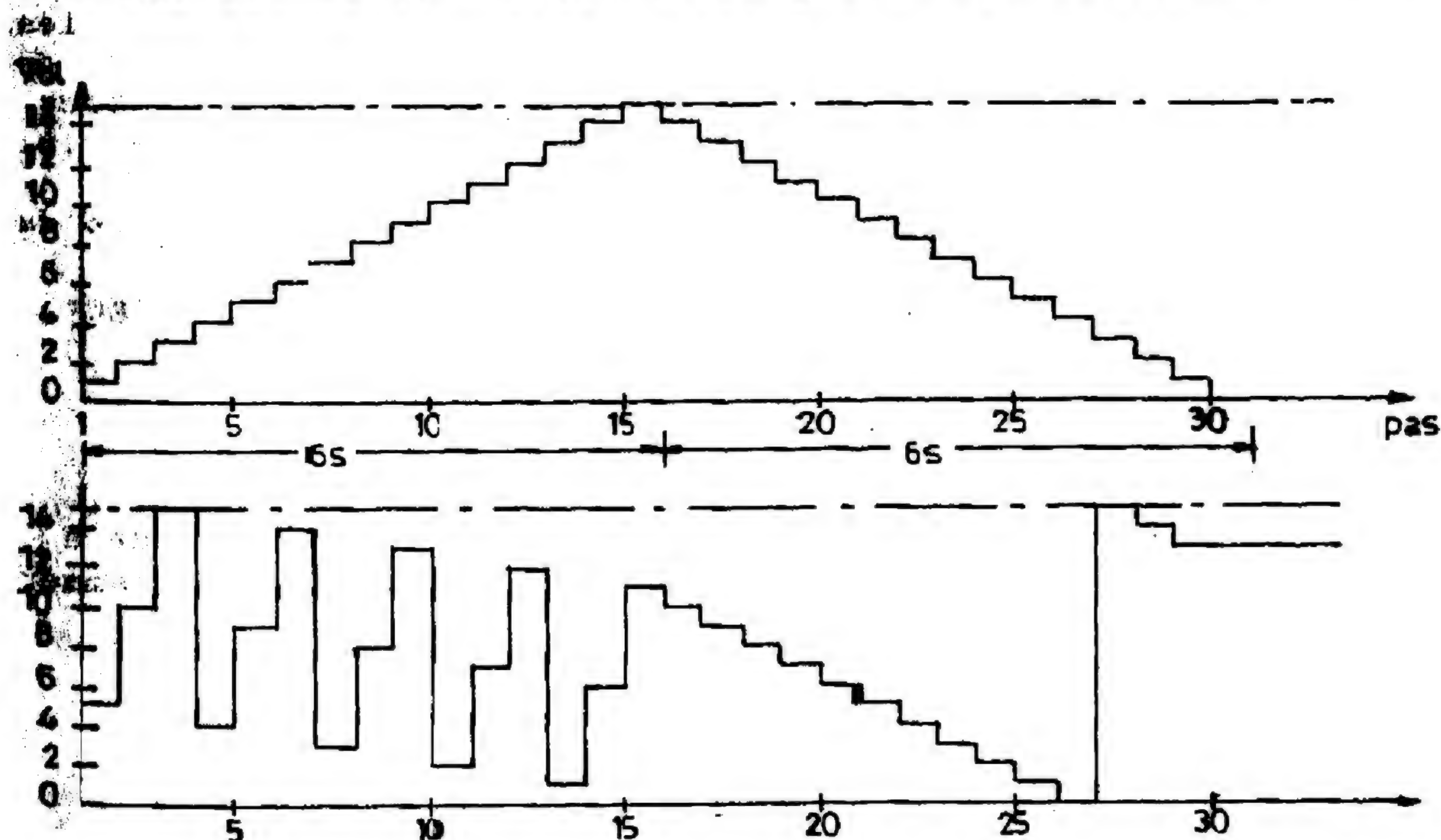


Schéma obtenu en remplaçant 10 par 10 ENV 1,15,5,40,15,-1,40

EXEMPLE N°2 : génération de coups de feu:

```

10 ENV 1,15,-1,6
20 SOUND 7,0,0,0,1,0,15
30 FOR I=1 TO 500:NEXT I
40 GOTO 20

```

- la ligne 10 définit l'enveloppe de volume N°1, c'est à dire que le volume part de son maximum 15 puis il est décrémenté de 1 toutes les 0,06 secondes (6*0,01).

- La ligne 20 définit le son de la manière suivante: le chiffre 7 sélectionne la sortie sur les canaux A,B et C, les trois zéros suivants signifient fréquence nulle, durée définie par l'enveloppe de volume et amplitude de départ 0. Le chiffre 1 donne le numéro de l'enveloppe et enfin, 15 définit la période de bruit.

B) programmation du PSG par CALL BD34.

EXEMPLE N°1 : génération d'un coup de feu.

Nous devons donc charger les registres du PSG avec les valeurs adéquates. L'accès au PSG n'étant pas immédiat, on utilise une routine de la ROM pour charger les registres. Avant d'appeler la routine, il est nécessaire de lui transmettre deux paramètres:

1- Le registre A du Z80 doit contenir le numéro du registre du PSG que l'on veut programmer.

2- Le registre C doit contenir la valeur à charger.

N° de bit	7	6	5	4	3	2	1	0
Val. déc.	128	64	32	16	8	4	2	1

registre du PSG									Valeur à programmer
R6	x	x	x	0	1	1	1	1	15
R7	x	x	0	0	0	1	1	1	7
R8	x	x	x	1	0	0	0	0	16
R9	x	x	x	1	0	0	0	0	16
R10	x	x	x	1	0	0	0	0	16
R11	0	0	0	0	0	0	0	0	0
R12	0	0	0	1	0	0	0	0	16
R13	x	x	x	x	0	0	0	0	0

R6 définit la période de bruit.

R7 valide la sortie du bruit sur les trois canaux A,B et C.

R8, R9 et R10 sélectionnent la modulation des 3 canaux au moyen du générateur d'enveloppe.

R11 permet l'ajustement fin de la période d'enveloppe.

R12 permet l'ajustement grossier de la période d'enveloppe.

R13 définit la forme de l'enveloppe (forme A).

Pour charger ces valeurs, nous devons utiliser un petit programme en assembleur:

```
3Exx      LD A,xx  (numéro du registre du PSG)
0Exx      LD C,xx  (valeur à charger)
CD 34 BD   CALL #BD34
C9         RET
```

Voici maintenant un petit programme BASIC qui vous permettra de charger les différents registres du PSG par la routine BD34:

```
10 MEMORY 29999
20 FOR I=30000 TO 30007
30 READ A$
40 POKE I,VAL("&" + A$)
50 NEXT I
60 DATA 3E,00,0E,00,CD,34,BD,C9
70 PRINT "ENTREZ LE NUMERO DU REGISTRE ";: INPUT R
80 POKE 30001,R
90 PRINT "ENTREZ LA VALEUR A CHARGER ";: INPUT V
100 POKE 30003,V
110 CALL 30000
120 GOTO 70
```

RUN

Charger les valeurs de R6 à R13.

Maintenant, chaque fois que vous introduisez R=13 et V=0, vous obtenez un coup de feu.

Pour changer la forme de l'enveloppe, il vous suffit de modifier la valeur de V (ex pour la forme C, R=13 et V=8).

EXEMPLE N°2: Génération d'un bruit de sirène.

Au programme de l'exemple 1, ajouter la ligne suivante:

```
15 PR=30001:PV=30003:PC=30000
```


Puis, remplacer les lignes 70 à 120 par:

```
70 POKE PR,7:POKE >V,62:CALL PC
80 POKE PR,8:POKE >V,15:CALL PC
90 FOR J=1 TO 3
100 POKE PR,0:POKE PV,I:CALL PC
110 NEXT I
120 FOR I=200 TO 100 STEP-1
130 POKE PR,0:POKE PV,I:CALL PC
140 NEXT I
150 NEXT J
160 POKE PR,8:POKE PV,0:CALL PC
```

La ligne 70 sélectionne le générateur sonore sur le canal A.

La ligne 80 force le canal A sur le volume maximum.

Les lignes 90 et 150 font varier la fréquence du canal A.

La ligne 160 arrête l'émission sonore sur le canal A.

EXEMPLE: Programme de chargement du PSG à partir d'une table en mémoire.

```
F5          PUSH AF
C5          PUSH BC
D5          PUSH DE
E5          PUSH HL
```

Sauvegarde des registres du Z80.

```
21 90 60    LD HL,#6090
```

Chargement dans le registre HL du pointeur de table.

```
E1          LE A,(HL)
```

Chargement dans l'accumulateur de la première valeur de la table. cette valeur doit être un numéro du registre.

```
FE FF      CP    0FF
CA 1F 60    JP    2,#601F
```

Test de la valeur de fin de table. Si le dernier élément chargé est égal à FF, le programme saute à l'adresse 601FH.

```
23          INC HI
```

On augmente le pointeur de table d'une unité.

```
4E          LD C,HL
```

Chargement dans le registre C du Z80 de la valeur à introduire dans le PSG.

```
CD 34 BD    CALL #BD34
```

Programmation du PSG par la routine BD34. A contient le

numéro du registre et C la valeur à charger.

```
0E FF    DELAY1  LD    C,#FF
06 FF                    LD    B,#FF
10 FE    DELAY2  DJN2  DELAY2
0D                    DEC    C
20 F9                    JR    NZ,DELAY1
```

Routine de temporisation avant de charger la valeur suivante dans le PSG.

```
23                    INC    HL
C3 07 60            JP    6007
```

tant que la valeur de fin de table n'est pas trouvée, le programme boucle.

```
E1                    POP    HL
D1                    POP    DE
C1                    POP    BC
F1                    POP    AF
C9                    RET
```

La valeur de fin de table étant trouvée, on restitue les registres du Z80 et on retourne au programme principal.

table origine: 6090 N° REGISTRE
6091 VALEUR A CHARGER
6092 N° REGISTRE
6093 VALEUR A CHARGER
....
....
.... FF valeur de fin de table.

programme Basic correspondant:

```
10 MEMORY &H5FFF
20 FOR I=&H6000 TO &H6023
30 READ A$
40 POKE I,VAL("&H"+A$)
50 NEXT I
60 DATA F5,C5,D5,E5,21,90,50,7E,FE,FF,CA,1F,60,23,
4E,CD,34,BD,0E,FF,06,FF,10,F3,0D,20,F9,23,C3,07,60,
E1,D1,C1,F1,C9
70 FOR I=&H6090 TO &H60C2
80 READ A
90 POKE I,A
100 NEXT I
110 DATA 7,62,3,15,0,239,0,213,0,190,0,179,0,159,0,
142,0,127,0,119,0,119,0,127,0,142,0,159,0,179,0,190,
0,213,0,239,0,190,0,159,0,119,0,159,0,190,0,239,0,
0,255
120 CALL &H6000
```


3.5 Les routines internes du générateur sonore.

La ROM de l'Amstrad possède toute une série de routines internes qui servent à la génération des sons et des effets sonores.

Nous poserons CE comme étant les conditions d'entrée de ces routines et CS, les conditions de sortie.

- BCA7** **RESET** du générateur.
Cette routine initialise le générateur sonore. Elle arrête tous les sons et efface toutes les queues d'attente.
CE: aucune.
CS: les registres AF,BC,DE et HL sont modifiés.
- BCAA** Ajoute un son dans une queue.
Le générateur pouvant traiter plusieurs sons sans s'occuper du processeur, cette routine a pour but de continuer le chargement des différents sons quand cela est possible.
CE: HL contient l'adresse du programme sonore qui doit se trouver dans les 32 K de mémoire vive centrale.
CS: si le son a pu être ajouté à la queue sonore, le sémaphore de carry est vrai et HL est modifié. Si toutes les queues sonores sont remplies et que le son n'a pu être ajouté à aucune d'entre elles, le sémaphore de carry est faux et HL est préservé. De toutes façons, AF,BC,DE et IX sont modifiés et les autres registres sont préservés.
- BCAD** Vérifie s'il y a de la place dans une queue.
CE: A contient le numéro du canal testé:
il vaut 1 si on veut tester le canal A,
il vaut 2 si on veut tester le canal B,
il vaut 4 si on veut tester le canal C.
CS: A contient l'état du canal testé:
B2 B1 B0: nombre de places libres dans la queue,
B3: rendez-vous avec le canal A
B4: rendez-vous avec le canal B,
B5: rendez-vous avec le canal C;
B6: attente au début de la queue,
B7: canal en train de jouer.

EMPLE: Voici un programme qui commande la fonction SOUND
rsqu'il n'y a pas de place dans la queue sonore du canal

```
10 MEMORY 29999
20 ENV 1,15,-1,5
30 FOR I=&H7500 TO &H7509
40 READ A$
50 POKE I,VAL("&H"+A$)
60 NEXT I
70 DATA 3E,02,CD,AD,BC,21,3A,75,77,C9
80 CALL 30000
90 A=PEEK(&H753A)
100 A=A AND 7
110 IF A=0 THEN GOTO 80
120 PRINT BIN$(A,8)
130 READ A
140 IF A=255 THEN GOTO 180
150 SOUND 2,A,0,0,1
160 GOTO 80
170 DATA 239,213,190,179,159,142,127,119,119,127,
142,159,179,190,213,239,159,95,60,255
180 PRINT"FIN"
```

Au début, la queue est vide et vous avez 5 places
libres. Les bits B2,B1 et B0 valent respectivement 1, 0 et
0, soit 4 en décimal. Après 5 passages, la queue est remplie
et les bits B2, B1 et B0 sont tous à 0.

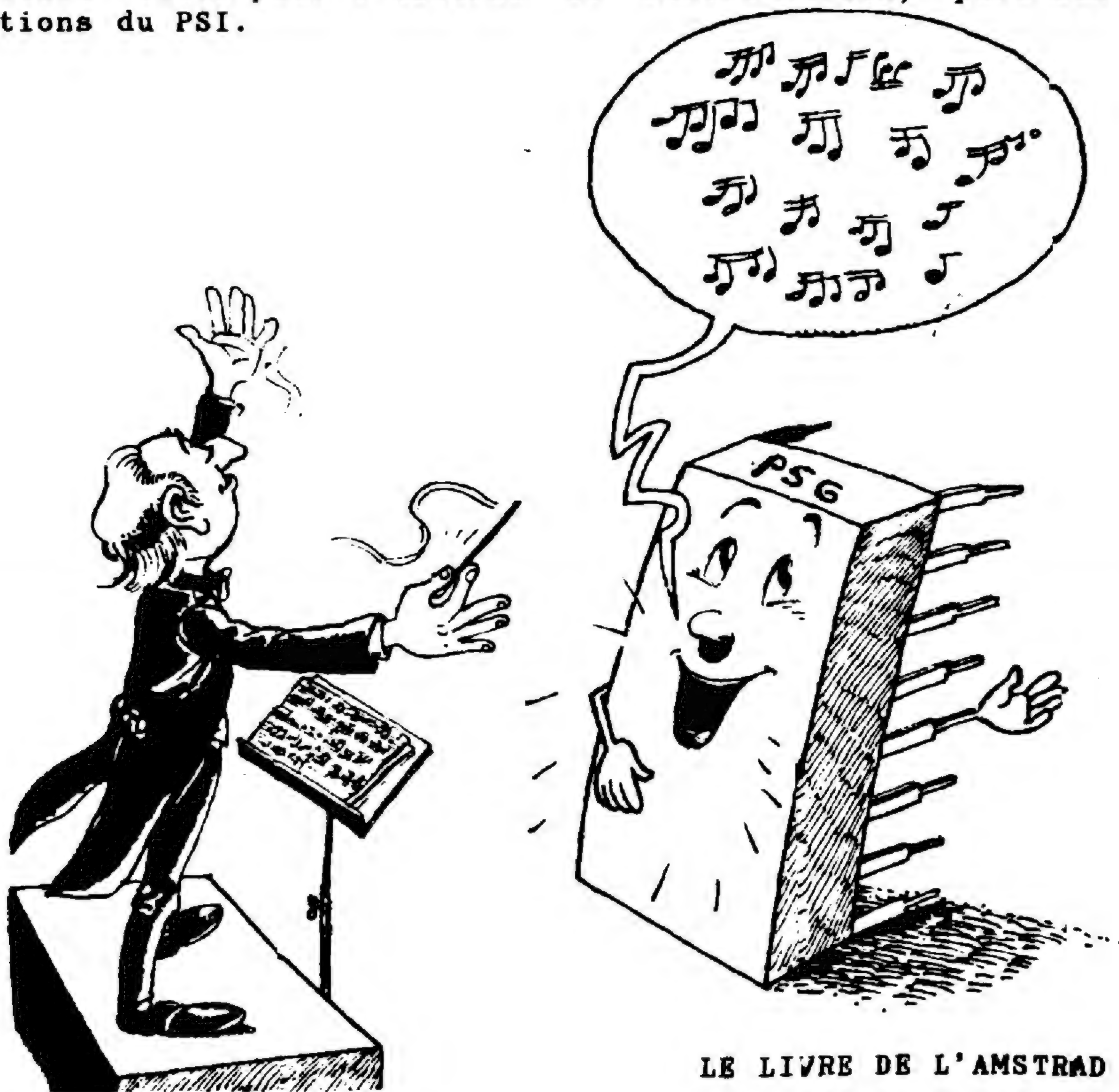
Programme en assembleur contenu dans le programme
basic:

3E 02	LD A,2
CD AD BC	CALL #BCAD
21 3A 75	LD HL,#753A
77	LD (HL),A
C9	RET

Autres routines:

BCB6	Arrêt de tous les sons.
BCB9	Démarre tous les sons arrêtés par la routine précédente.
BCB0	Prépare l'exécution d'une interruption lorsqu'une queue sonore sera vide.
BCB3	Permet de rétablir les sons arrêtés sur chaque canal.
BCBC	Etablit une des 15 enveloppes d'amplitude programmable.
BCBF	Etablit une des 15 enveloppes de fréquence programmable.
BCC2	Fournit l'adresse d'une enveloppe d'amplitude.
BCC5	Fournit l'adresse d'une enveloppe de ton.

Si le lecteur désire de plus amples informations au sujet de ces routines, nous l'invitons à consulter l'ouvrage intitulé "Clefs pour l'Amstrad" de Daniel Martin, paru aux éditions du PSI.



L'INTERFACE PERIPHERIQUE PPI 8255A.

4.1 Généralités.

Le PPI est un circuit fabriqué par INTEL sous la dénomination 8255A. C'est un circuit d'interface prévu pour les processeurs de la famille du 8080. Il est pourvu de 24 bits d'entrée/sortie qui peuvent être programmés individuellement en deux groupes de 12 bits et utilisés dans trois modes principaux.

Dans le premier mode (MODE 0), chaque groupe de 12 bits peut être programmé en tranches de 4 bits en entrée ou en sortie.

Dans le second mode (MODE 1), chaque groupe de 12 bits peut être programmé pour obtenir 8 lignes ou les 8 bits sont en entrée/sortie, les 4 autres lignes sont utilisées pour le HANDSHAKING (contrôle de transmission) et les signaux d'interruption.

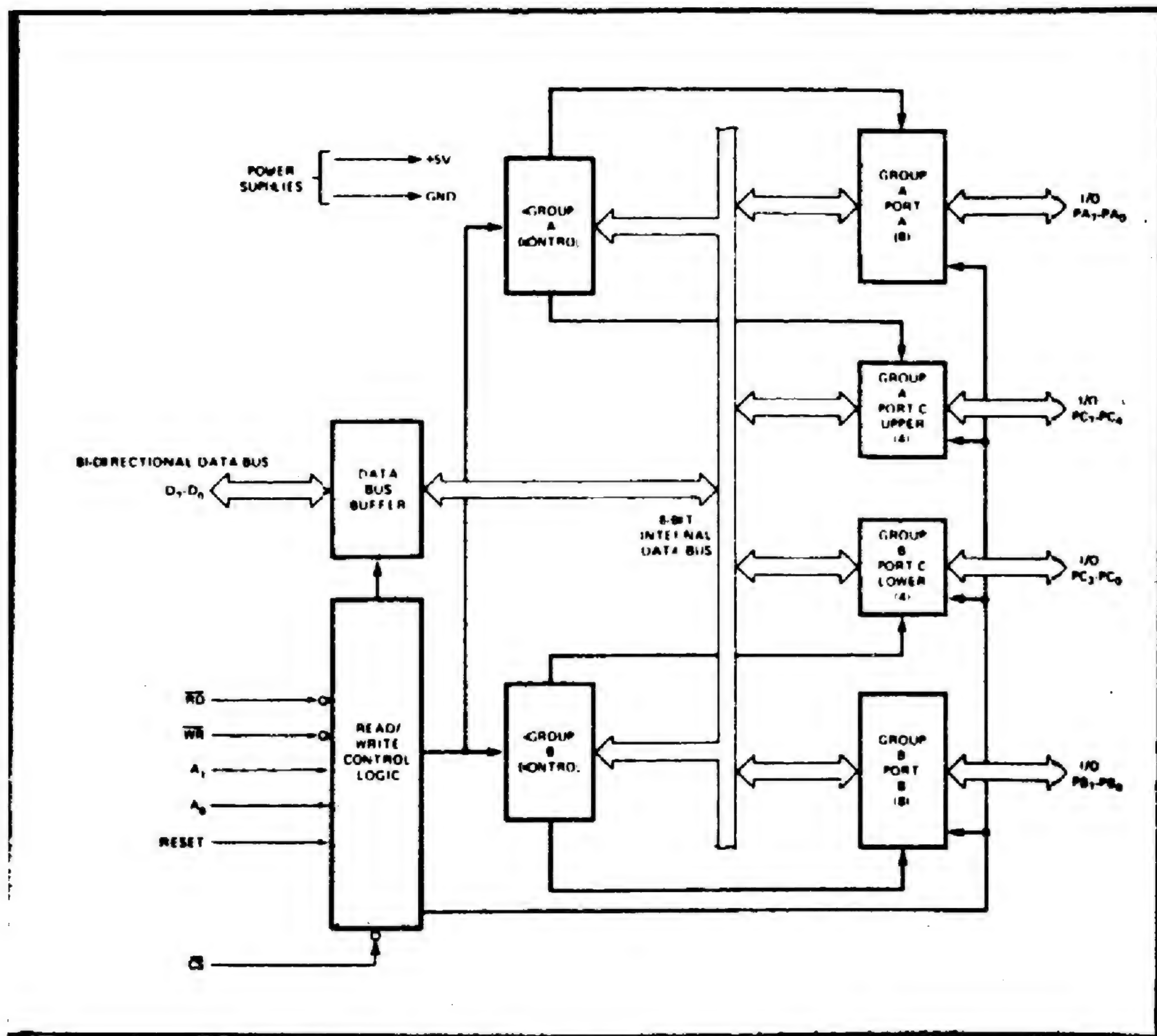
Le troisième mode (MODE 2), est un mode bidirectionnel qui utilise 8 bits en entrée/sortie et 5 bits pour le contrôle de transmission (HANDSHAKING).

Le PPI possède donc la possibilité de positionner les lignes de sortie à un état logique haut (bit à l'état 1), ou bas (bit à l'état 0).

Pour plus de clarté, le PPI est un circuit d'interface parallèle composé de 3 ports de 8 bits distincts appelés PORT A, PORT B et PORT C.

Le port C se divise en deux groupes de 4 bits pour former avec les ports A et B deux groupes de 12 bits.

SCHEMA BLOC.



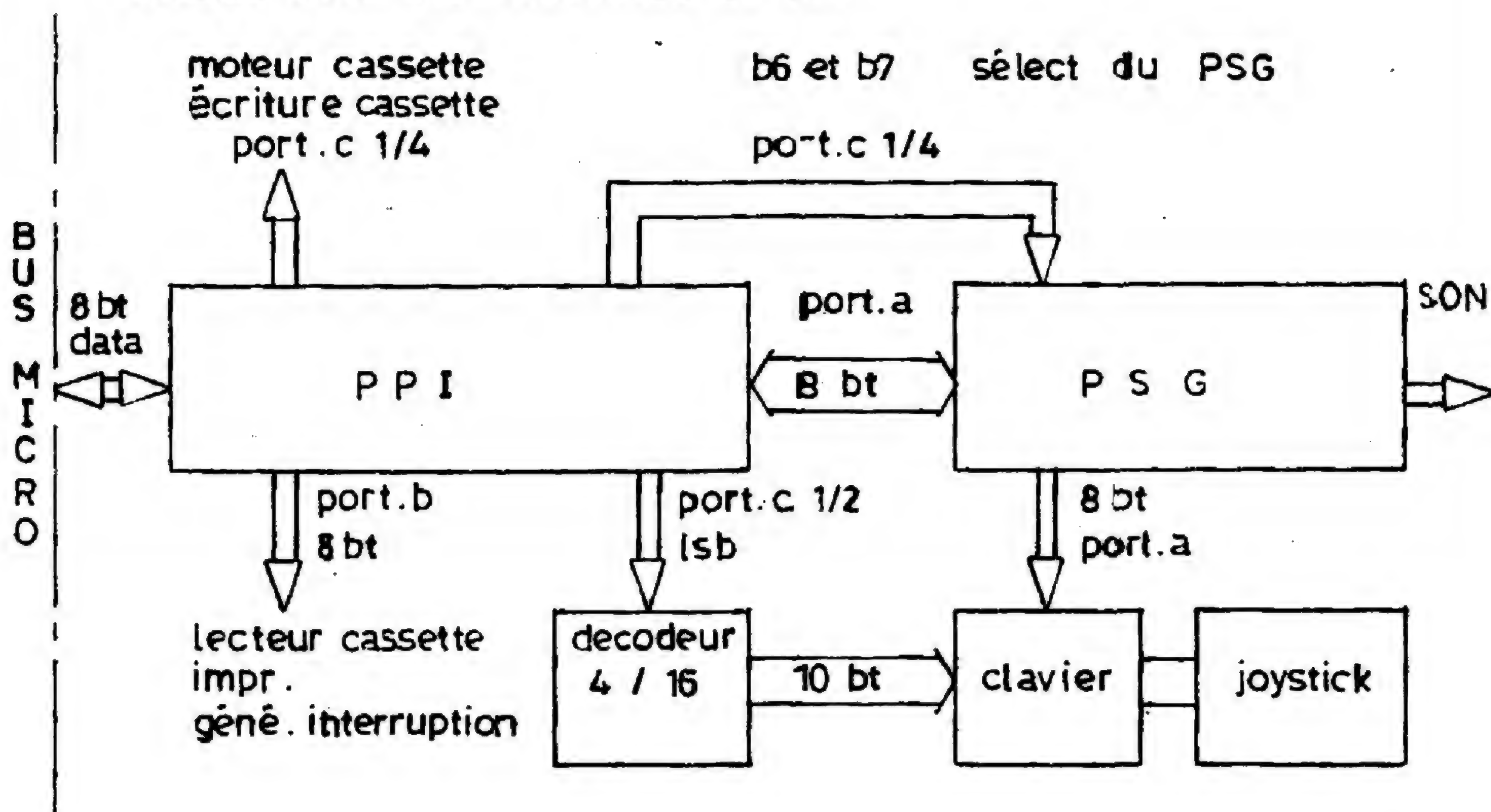
4.2 Découpage et utilisation des ports A, B et C.

4.2.1 INTRODUCTION.

Le PPI joue un rôle très important dans la configuration interne (hardware) de l'ordinateur. En effet, il a pour mission la gestion du clavier et des manettes de jeux, la gestion du signal "occupé" (busy) de l'imprimante, du lecteur de cassettes, de l'horloge d'interruption et de la programmation du PSG.

Toutes ces fonctions ne se trouvent pas directement interfacées sur le bus de données du microprocesseur. Seuls les 8 bits de données du PPI se trouvent en communication directe avec le bus des données du Z80. Cette organisation permet un gain de place non négligeable en mémoire.

Schéma bloc des fonctions du PPI:



4.2.2 LE PORT A.

Le port A est certainement le plus important car c'est lui qui permet la lecture du clavier et des manettes de jeux par l'intermédiaire du registre R14 du PSG.

Le Port A est donc utilisé pour interfacer le bus interne de l'AY3-8912. Il permet de sélectionner les adresses des différents registres du PSG ainsi que le chargement de leurs valeurs.

Les échanges entre le PPI et le PSG pouvant s'effectuer dans les deux sens, le port A du PPI sera utilisé en bidirectionnel, c'est à dire, en entrée et en sortie.

REMARQUE: Le port A du PSG est uniquement utilisé pour la lecture du clavier et des manettes de jeux. C'est pour cette raison que le bit 6 du registre 7 du PSG doit toujours valoir 0.

Définition des bits du Port A:

BIT 7 :	DATA/ADRESSE	PA7	connecté	au	PSG
BIT 6 :	" " "	PA6	" "	" "	" "
BIT 5 :	" " "	PA5	" "	" "	" "
BIT 4 :	" " "	PA4	" "	" "	" "
BIT 3 :	" " "	PA3	" "	" "	" "
BIT 2 :	" " "	PA2	" "	" "	" "
BIT 1 :	" " "	PA1	" "	" "	" "
BIT 0 :	" " "	PA0	" "	" "	" "

4.2.3 LE PORT B.

Le port B est utilisé pour lire les données en provenance du lecteur de cassettes.

Il permet également de tester si l'imprimante connectée sur l'entrée parallèle (centronics) est en fonctionnement. Le signal émis par celle-ci est appelé signal Busy (occupé).

Il sert aussi pour l'acquisition du signal horloge. L'horloge est fournie par les pulses du retour trame du signal vidéo. Elle sert de générateur d'interruptions.

Le bit 4 du port B sert à la sélection de la fréquence du réseau 60 ou 50 hertz.

Le bit 5 indique si une extension est connectée à l'ordinateur.

Enfin, les bits B3, B2 et B1 permettent de sélectionner le nom de l'ordinateur que vous possédez.

Si vous ouvrez votre ordinateur, vous remarquerez en haut à gauche quatre pontages: LK1, LK2, LK3, LK4 qui correspondent aux bits B0, B1, B3 et B4.

Si vous effectuez les pontages suivants, vous obtenez:

LK1	LK2	LK3	
0	0	0	AMSTRAD
0	0	1	ORION
0	1	0	SCHNEIDER
0	1	1	AWA
1	0	0	SOLAVOX
1	0	1	SAIPHO
1	1	0	TRIUMPH
1	1	1	ISP

NOTE un 1 correspond à un pontage.

Définition des bits du port B:

BIT 7 : signal de lecture des données sur cassette.
BIT 5 : signal BUSY sur sortie Centronics.
BIT 5 : extension active.
BIT 4 : sélection 50 ou 60 hertz.
BIT 3 \
BIT 2 : choix du nom de l'ordinateur.
BIT 1 /
BIT 0 : pulse de retour TRAM.

Le port B est uniquement programmé en entrée.

4.2.4 LE PORT C.

Le port C a la direction de 4 fonctions: la première consiste en la gestion du mode de fonctionnement du PSG, la seconde, en la sélection des lignes de la matrice clavier, la troisième, en l'écriture des données sur la cassette et la quatrième, en la gestion du contrôle ON/OFF du moteur d'entraînement du lecteur de cassette.

A) Mode de fonctionnement du PSG, Bit 7 et Bit 6.

Lors de l'étude du PSG, nous avons omis volontairement de parler des trois signaux importants du PSG qui sont BDIR, BC1 et BC2.

Ces trois signaux permettent le contrôle des entrées/sorties sur le bus interne du PSG. BC2 étant forcé au +5 v, les commandes possibles avec BDIR et BC1 sont au nombre de 4 (2 esp 2).

Décimal	BDIR	BC2	BC1
0	0	1	0
1	0	1	0
2	1	1	0
3	1	1	1

Valeur 0 : rend le PSG inactif

Valeur 1 : permet de lire le PSG

Valeur 2 : permet d'écrire dans le PSG

Valeur 3 : permet de charger une adresse d'un des registres du PSG.

B) Sélection des lignes du clavier: B3 à B0.

Les bits B3 à B0 permettent via un décodeur 4 vers 16 de sélectionner parmi les 16 lignes utilisées de la matrice une et une seule ligne pour la lecture du clavier.

C) Ecriture sur la cassette.

Le bit B5 du port C écrit une suite de 1 et de 0 qui seront utilisés pour l'écriture des données sur le lecteur de cassettes.

D) Contrôle moteur du lecteur de cassettes.

Le bit B4 à l'état 1 commande le démarrage du moteur du lecteur tandis que l'état 0 de ce même bit en provoque l'arrêt.

Définition des bits du port C:

BIT 7 : BDIR du PSG

BIT 6 : BC1 du PSG

BIT 5 : écriture des données sur la cassette

BIT 4 : commande du moteur ON/OFF

BIT 3 : sélection des lignes du clavier SCL3

BIT 2 : sélection des lignes du clavier SCL2

BIT 1 : sélection des lignes du clavier SCL1

BIT 0 : sélection des lignes du clavier SCL0

4.3 Programmation du PPI.

4.3.1 INTRODUCTION.

Le PPI est interfacé aux adresses suivantes:

- A) F4xx lecture et écriture du port A
- B) F5xx lecture du port B
- C) F6xx écriture du port C
- D) F7xx écriture du registre de contrôle

REMARQUE: De la configuration matérielle, nous pouvons conclure que le port A est utilisé en entrée/sortie, le port B uniquement en lecture, et le port C, en écriture. Parmi les modes décrits brièvement dans les généralités, seul le mode 0 sera étudié car il suffit à toutes les manipulations envisagées.

Le PPI est programmable au travers d'un registre de contrôle dans lequel on ne peut qu'écrire. Aucune lecture de ce registre n'est permise.

Les ports du PPI sont divisés en deux groupes. Le groupe A, composé du port A et des 4 bits de poids fort du port C; et le groupe B, composé du port B et des 4 bits de poids faible du port C.

4.3.2 PROGRAMMATION.

L'écriture ou la lecture d'un des ports du PPI se fait par une simple instruction:

OUT ADRESSE, DATA ou INP(ADRESSE)

sur une des adresses qui lui sont réservées.

Ecriture du registre de contrôle adresse F7xx:

Le mot de contrôle est un mot de 8 bits. Voici la signification de chacun d'entre eux:

- BIT 7 : toujours 1 si c'est un mot de contrôle.
- BITS 6 et 5 : déterminent le mode de fonctionnement du groupe A. Pour sélectionner le mode 0, les bits doivent se trouver à l'état logique 0 (voir figure).
- BIT 4 : Il détermine le sens de fonctionnement du port A. Un 1 signifie que le port A est en entrée et un 0, qu'il est en sortie.
REM: ne pas confondre PORT A avec GROUPE A.
- BIT 3 : détermine le sens de fonctionnement de la partie haute du port C, c'est à dire les 4 bits de poids fort faisant partie du groupe A du PPI.
- BIT 2 : Détermine le mode de fonctionnement du groupe B. 0 signifie mode 0 et 1, mode 1. Il sera toujours à 0.
- BIT 1 : détermine le sens de fonctionnement du port B, 0 signifie que le port B est en sortie et 1, en entrée. Il sera toujours à 1.
- BIT 0 : détermine le sens de fonctionnement de la partie basse du port C. 0 signifie en sortie et 1, en entrée.

TABLEAU RECAPITULATIF : MODE 0 DU PPI.

A				G R O U P E A			G R O U P E B	
D4	D3	D1	D0	PORT A	PORT C haut	#	PORT B	PORT C bas
0	0	0	0	SORTIE	SORTIE	0	SORTIE	SORTIE
0	0	0	1	SORTIE	SORTIE	1	SORTIE	ENTREE
0	0	1	0	SORTIE	SORTIE	2	ENTREE	SORTIE
0	0	1	1	SORTIE	SORTIE	3	ENTREE	ENTREE
0	1	0	0	SORTIE	ENTREE	4	SORTIE	SORTIE
0	1	0	1	SORTIE	ENTREE	5	SORTIE	ENTREE
0	1	1	0	SORTIE	ENTREE	6	ENTREE	SORTIE
0	1	1	1	SORTIE	ENTREE	7	ENTREE	ENTREE
1	0	0	0	ENTREE	SORTIE	8	SORTIE	SORTIE
1	0	0	1	ENTREE	SORTIE	9	SORTIE	ENTREE
1	0	1	0	ENTREE	SORTIE	10	ENTREE	SORTIE
1	0	1	1	ENTREE	SORTIE	11	ENTREE	ENTREE
1	1	0	0	ENTREE	ENTREE	12	SORTIE	SORTIE
1	1	0	1	ENTREE	ENTREE	13	SORTIE	ENTREE
1	1	1	0	ENTREE	ENTREE	14	ENTREE	SORTIE
1	1	1	1	ENTREE	ENTREE	15	ENTREE	ENTREE

Si le bit 7 du registre de contrôle est égal à 0, le registre n'est plus utilisé en tant que contrôleur des ports mais il permet de positionner un par un les bits du port C.

Dans ce mode de fonctionnement, les bits 6, 5 et 4 du registre ne sont pas utilisés. Les bits 3, 2 et 1 donnent le numéro du bit à positionner ($2^{\text{exp } 3} = 8$).

Le bit 0 donne la valeur du bit que l'on veut positionner. 1 signifie que l'état logique du bit sera 1 et 0 que l'état logique sera 0.

EXEMPLE 1: pour positionner le bit 6 du port C à 1, il faut charger le registre de contrôle avec la valeur 11, c'est à dire: B3, B2, B1 et B0 doivent valoir respectivement 1, 1, 0 et 1.

EXEMPLE 2: démarrage du moteur du lecteur de cassettes par programmation du registre de contrôle. Il faut positionner le bit 4 du port C à 1:

```
10 OUT &HF700,&X00001001
```

Analysons l'octet	B7	B6	B5	B4	B3	B2	B1	B0
	0	0	0	0	1	0	0	1

Le premier bit en partant de la gauche (B7) vaut 0. Cela signifie que le registre n'est plus en mode contrôle mais bien en mode de positionnement de bits. Les trois bits suivants (B6 à B4) ne servent à rien. Les trois bits suivants (B3 à B1) valent respectivement 1, 0 et 0, c'est à dire 4 en décimal. C'est donc le bit 4 du port C qui est sélectionné. Le bit 0 vaut 1, ce qui signifie que le bit 4 prend la valeur logique 1.

Une autre façon de procéder est d'utiliser directement le port C à l'adresse F6xx:

```
10 OUT &HF600,&X00010000 (binaire)
ou 10 OUT &HF600,&H10 (hexadécimal)
ou 10 OUT &HF600,16 (décimal)
commande le démarrage du lecteur de cassettes.
```

```
10 OUT &HF600,0
arrête ce même moteur.
```

EXEMPLE 3: lecture du port B.

```
10 PRINT BIN$(INP(&HF500),8)
20 GOTO 10
```


4.4 Progratipn du PSG au travers du PPI.

Comme nous l'avons vu précédemment, le PSG n'est pas directement en contact avec le bus du microprocesseur, mais bien via le port A bidirectionnel du PPI. La programmation du PSG doit donc s'effectuer par ce dernier. Cette opération se déroule en 4 phases:

- 1) Introduction sur le port A du PPI de l'adresse du numéro du registre PSG à programmer.
- 2) Introduction sur les deux bits de poids fort du port C, B7 et B5, la valeur 11. Ces deux bits correspondent à la sélection PSG en mode adressage. A ce stade, l'adresse du registre PSG est chargée.
- 3) Introduction sur le port A du PPI la valeur à charger dans le registre sélectionné.
- 4) Sélectionner le PSG en mode écriture en introduisant sur les bits 6 et 7 du port C la valeur 10.

Pour en comprendre la procédure utilisée, regardons ensemble la routine interne BD34 qui s'occupe de la programmation du PSG:

Si nous effectuons la lecture des octets contenus dans la RAM à partir de l'adresse BD34:

```
10R I=&HBD34 TO &HBD36
20INT HEX$(PEEK(I));" ";
30XT I
```

nous obtenons:

```
BI    CF    RST 8
BI    26 (CPC464) ou 53 (CPC664)
BI    88
```

La première ligne est un restart en adresse 0008H. Les octets de ces deux adresses suivantes BD35 et BD36 donnent la valeur 88H (CPC464) ou 8853H (CPC664). Les bits 7 et 6 de BD36 donnent le numéro de la ROM où se situe la routine. BD34 se trouve donc en ROM 1, c'est à dire ROM basse, à l'adresse 26H (CPC464) ou 0853H (CPC664).

Pour pouvoir lire le contenu de la ROM, nous devons:

- 1) commuter la ROM basse: CALL #B906
- 2) copier le contenu de la ROM dans la RAM.

Le petit programme suivant permet de copier la ROM en mémoire vive et d'afficher le contenu de la routine BD34 en hexadécimal.

Programme Basic:

```
10 MEMORY &H5000
20 FOR I=&H5001 TO &H500F
30 READ A$
40 POKE I,VAL("&H"+A$)
50 NEXT I
60 DATA CD,06,B9,21,00,00,11,00,60,01,FF,3F,ED,B0,C9
65 CALL &5001
70 FOR I=&H6826 TO &H6846 : REM CPC 464
80 PRINT HEX$(PEEK(I));" ";
90 NEXT I
```

RUN

CPC664 Remplacer ligne 70 : 6826 par 6853 et 6846 par 6873

Programme en assembleur contenu dans le programme basic:
copie de la ROM en RAM:

CD 06 B9	CALL #B906
21 00 00	LD HL,00
11 00 60	LD DE,#6000
01 FF 3F	LD BC,#3FFF
ED B0	LDIR
C9	RET

La première ligne commute la ROM basse. Les lignes 2, 3, 4 et 5 transfèrent un bloc de 3FFFH octets à partir de l'adresse 0 vers l'adresse 6000H.

Listing résultant du programme

```
F3 06 F4 ED 79 06 F6 ED 78 F6 CD ED 79 E6 3F ED
79 06 F4 ED 49 06 F6 4F F6 80 ED 79 ED 49 FB C9
```


Analyse complète du contenu de la routine BD34

RAPPEL: CE de BD34 registre A=numéro du registre
C=valeur à charger

nous obtenons:

F3 DI interdiction des interruptions.

06 F4 LD B,#F4
ED F9 OUT (C),A

place sur les adresses de poids fort F4. Sélectionne le port A du PPI à l'adresse F4xx, avec le registre A du Z80 qui contient le numéro du registre du PSG.

06 F6 LD B,#6F6
ED 78 IN A,(C)

Lecture du port C du PPI. Les bits 7 et 6 donnent l'état du PSG: 00 = non actif

01 = lecture

10 = écriture

11 = chargement d'adresse

F6 C0 OR #C0

Positionne les bits 6 et 7 à 11:

	B7	B6	B5	B4	B3	B2	B1	B0
A =	x	x	x	x	x	x	x	x
C0=	1	1	0	0	0	0	0	0

OR:	1	1	x	x	x	x	x	x

ED 79 OUT (C),A

A ce stade, l'adresse du registre du PSG est chargée dans celui-ci

E6 3F AND 3F
ED 79 OUT (C),A

Positionne les bits 7 et 6 à 00. Le PSG est inactif.

A =	1	1	x	x	x	x	x	x
3F =	0	0	1	1	1	1	1	1

AND	0	0	x	x	x	x	x	x

06 F4 LD B,#F4
ED 49 OUT (C),C

Chargement sur le port A du PPI de la valeur du registre sélectionné.

06 F6 LD B,0F6H

Sélectionne le port C pour la commande du PSG.

4F LD C,A
sauvegarde du registre A

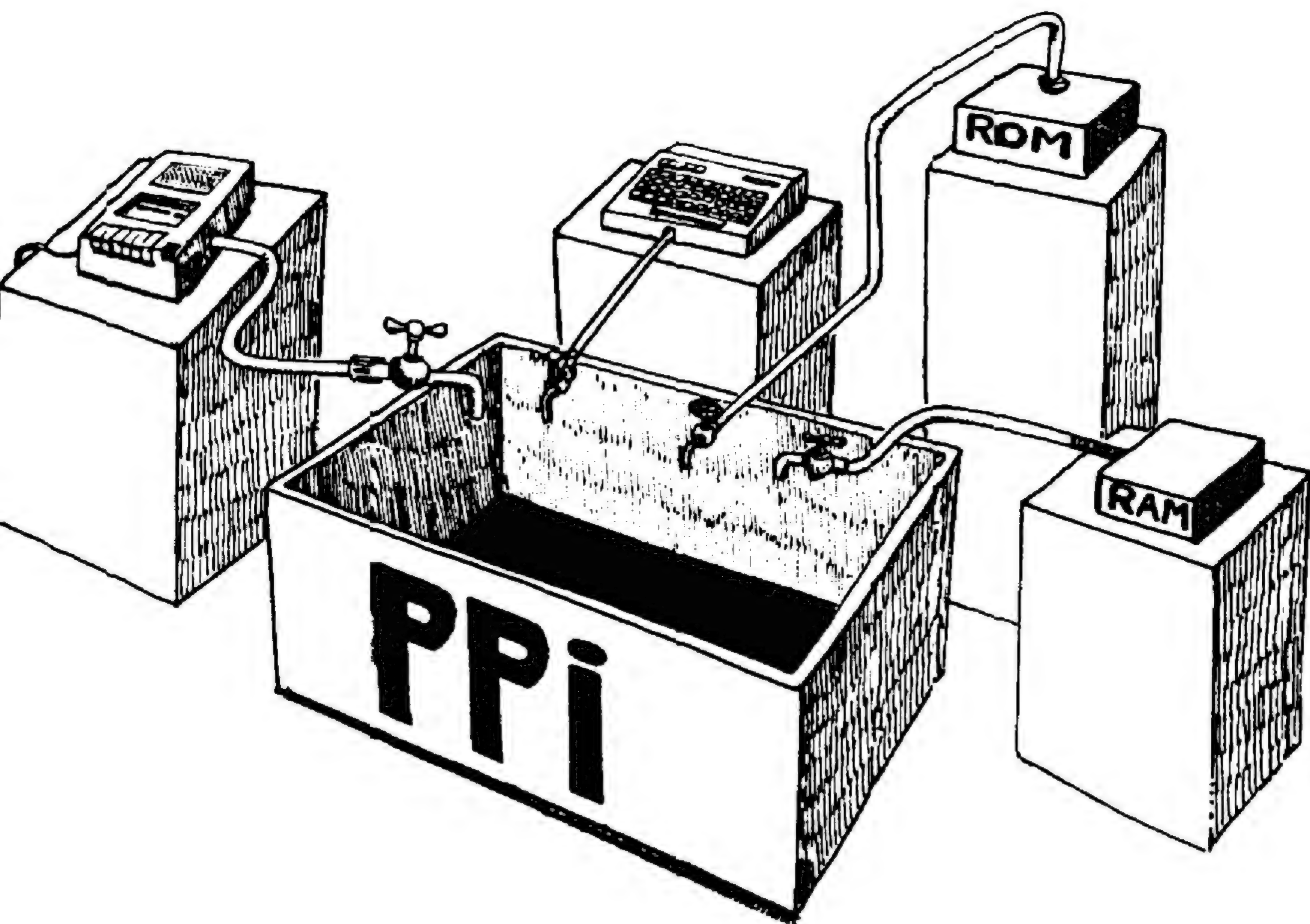
F6 80 OR #80
ED 79 OUT (C),A
Sélectionne le PSG en mode écriture B7=1 et B6=0

A = 0 0 x x x x x x
80 = 1 0 0 0 0 0 0 0

OR 1 0 x x x x x x

ED 49 OUT (C),C
Sortie sur le port C: sélectionne le PSG en mode inactif.
B7=0 et B6=0.

FB EI autorisation des interruptions
C9 RET retour au programme principal.



4.5 Lecture du clavier au travers du PPI.

4.5.1 DESCRIPTION DU CLAVIER.

Le clavier de l'Amstrad, lui non plus, n'est pas directement en contact avec le bus du microprocesseur. Il se compose de huit colonnes et de dix lignes. Les 8 colonnes sont connectées au port A du PSG, lui-même connecté au port A du PPI. Les 10 lignes proviennent du décodage 4 vers 16 des 4 bits de poids faible du port C du PPI (B3 à B0) dont seules les valeurs de 0 à 9 sont utilisées.

Le clavier est donc comparable à une matrice de 8 colonnes et de 10 lignes. Elle permet de décoder 80 touches disposées selon le schéma suivant:

PORT C DU PPI				D E C O D E R	PORT A DU PSG, REGISTRE R14							
B3	B2	B1	B0	U	B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	P.	pCR	pF3	pF6	pF9	p↓	p→	p↑
0	0	0	1	1	pF0	pF2	pF1	pF5	pF8	pF7	pcopy	p←
0	0	1	0	2	CTRL	\	SFT	pF4	}	CR	[CLR
0	0	1	1	3	>	?	*	+	P	@	=	£
0	1	0	0	4	<	M	K	L	I	O	9	0
0	1	0	1	5	SPC	N	J	H	Y	U	7	8
0	1	1	0	6	V	B	F	G	T	R	5	6
0	1	1	1	7	X	C	D	S	W	E	3	4
1	0	0	0	8	Z	capsl	A	tab	Q	esc	2	1
1	0	0	1	9	del			Jtir	J→	J←	J↓	J↑

REMARQUE: touche précédée de p signifie petit clavier
touche précédée de J signifie Joystick
touche SFT signifie SHIFT
touche CAPSL signifie CAPS LOCK
touche SPC signifie SPACE BARRE

4.5.2 LECTURE DU CLAVIER.

Pour lire la matrice clavier par le PPI, il faut procéder comme suit:

- 1) charger le registre R7 du PSG de manière à y positionner le bit B6 à 0. Après cette opération, la partie A du PSG se trouve en mode input (entrée).
- 2) Introduire dans le PSG l'adresse du registre R14, qui est le registre du port A du PSG. C'est dans ce registre que l'on trouvera la valeur qui sera lue ultérieurement.
- 3) Introduire sur les bits de poids faible (B3, B2, B1 et B0) du port C du PPI la valeur correspondant au numéro de la ligne du clavier à lire.
- 4) Effectuer la lecture du port A du PSG
- 5) Lire sur le port A du PPI la valeur du port A du PSG.

A ce moment, les 8 bits du port A du PSG contenant la valeur de la ligne clavier sélectionnée se trouvent dans le registre A du Z80, c'est à dire, dans l'accumulateur.

Pour plus de clarté, voici un petit programme en assembleur qui permet de lire la ligne 0 du clavier:

1°) programme Basic:

```
10 MEMORY &H6000
20 FOR I=&H6001 TO &H6027
30 READ A$
40 POKE I,VAL("&H"+A$)
50 NEXT I
60 DATA 3E,07,0E,00,CD,34,BD,06,F4,3E,0E,ED,79,06,
        F6,3E,C0,ED,79,06,F7,3E,92,ED,79,06,F6,3E,
        40,ED,79,06,F4,ED,78,32,80,06,C9
70 CALL &H6001
80 PRINT BIN$(PEEK(&H6080),8)
90 GOTO 70
```

RUN

2°) programme assembleur contenu dans le programme Basic:

```
3E 07          LD A,7
0E 00          LD C,0
CD 34 BD      CALL #BD34
```

Les trois premières lignes chargent dans le registre R7 du PSG la valeur 0. Le bit 6 du PSG est donc bien positionné à 0. On utilise la routine BD34 développée au paragraphe (5.4) pour charger le PSG. A contient le numéro du registre du PSG

à programmer et C, la valeur à introduire.

```
06 F4      LD B,#F4
3E 0E      LD A,#0E
ED 79      OUT (C),A
```

Ces trois lignes suivantes chargent sur le port A du PPI l'adresse du registre R14 du PSG, qui est le port A du PSG. Nous rappelons au passage que OUT (C),A charge le contenu du registre B sur les adresses hautes du Z80.

```
06 F6      LD B,#F6
3E C0      LD A,#C0
ED 79      OUT (C),A
```

Ces trois lignes chargent la valeur C0 sur le port C du PPI. En binaire, C0 s'écrit: 1100 0000. Les bits B7 et B6 sont donc à l'état 1. Ils sélectionnent le PSG en mode adressage. A ce stade, l'adresse du registre R14 du PSG (14 décimal ou 0E hexa) est introduite dans le PSG.

```
06 F6      LD B,#F7
3E 92      LD A,#92
ED 79      OUT (C),A
```

Au début du programme, le port A du PPI est en mode sortie. Pour pouvoir procéder à la lecture du registre R14 du PSG, il doit être en entrée. Ces trois lignes modifient le registre de contrôle du PPI pour sélectionner le port A en entrée, le port C en sortie et le port B en entrée (voir 5.3.2).

```
06 F3      LD B,#F6
3E 40      LD A,#40      (*)
ED 79      OUT (C),A
```

Le port A du PPI étant maintenant en entrée, on peut effectuer la lecture du registre R14 du PSG. Ces trois lignes de programme chargent sur le port C du PPI la valeur hexadécimale 40H, soit en binaire, 0100 000. Les bits de poids fort B7 et B6 valent respectivement 0 et 1. Cette valeur sélectionne le PSG en mode lecture. Les 4 bits de poids faible (B3 à B0) ont la valeur 0. Ils sélectionnent la ligne 0 de la matrice clavier. Dès cet instant, la valeur du registre R14 du PSG se trouve sur le bus du PSG.

```
06 F4      LD B,#F4
ED 78      IN A,(C)
```


Ces deux lignes ont pour but de lire sur le port A du PPI la valeur du registre R14. Le registre A du Z80 contient la valeur de la ligne clavier sélectionnée. Un des bits de l'accumulateur est à 0 si la touche correspondante est enfoncée, sinon il vaut 1.

```
32 80 60      LD(#6080),A
```


Les deux dernières lignes sauvent l'accumulateur du Z80 à une adresse mémoire, en l'occurrence à l'adresse 6080H dans notre cas, puis retourne au programme principal.

Maintenant quand vous poussez sur une des touches de la ligne 0 du clavier, vous voyez apparaître sur votre écran un 0 correspondant à la touche enfoncée. Vous pouvez, par la même méthode, décoder n'importe quelle ligne du clavier. Il suffit pour cela de changer dans le programme le numéro de sélection de la ligne sur le port C (*). 40H = ligne 0, 41H = ligne 1, ..., 49H = ligne 9.

De même, vous pouvez également tester l'appui d'une touche précise sans passer par l'INPUT ou l'INKEY\$.

EXEMPLE: Si vous désirez tester la touche  du petit clavier, il suffit de remplacer la ligne 80 du programme Basic par les lignes:

```
80 A=PEEK(&H6080) AND &X00000001
81 IF A=0 THEN GOTO 100
90 GOTO 70
100 PRINT "OK"
```


4.6 La manette de jeux (Joystick).

La manette de jeux de l'Amstrad fait partie intégrante du clavier de l'ordinateur. Sa lecture suit donc exactement les mêmes directives que celles énoncées pour la lecture du clavier.

Pour lire la manette de jeux, il suffit de modifier dans le programme le numéro de sélection de la ligne du clavier 40H par 49H. Dès lors, vous pouvez tester les 5 fonctions de la manette de jeux.

EXEMPLE: Remplacer à la ligne 60 du programme Basic précédent 40 par 49 et ajouter les lignes suivantes:

```
80 A=PEEK(&H6080)
81 IF A=254 THEN PRINT "HAUT"
82 IF A=253 THEN PRINT "BAS"
83 IF A=251 THEN PRINT "GAUCHE"
84 IF A=247 THEN PRINT "DROITE"
85 IF A=239 THEN PRINT "FET"
86 A=A AND &X100000000
87 IF A=0 THEN GOTO 100
90 GOTO 70
100 PRINT "FIN"
```

RUN

Pour sortir de ce programme, enfoncer la touche DEL.

REMARQUE: La valeur contenue dans le registre A du Z80 ne correspond pas à l'affichage de la touche, mais bien à sa position dans la matrice clavier. Pour pouvoir être affichée, le contenu de la matrice doit être analysé et traité. Ceci fera l'objet du prochain chapitre.

CHAPITRE V

LES PERIPHERIQUES EN CONTACT AVEC LE PPI.

5.1 Le lecteur de cassettes.

5.1.1 GENERALITES

La cassette est certainement, pour les ordinateurs familiaux comme l'AMSTRAD, le moyen le plus économique permettant d'archiver les programmes et les données.

Un grand nombre de ces ordinateurs est livré avec soit un lecteur incorporé, soit une prise pour lecteur externe. Ceci évite au débutant l'achat d'un lecteur de disquette dont le prix est généralement élevé par rapport au prix de base de l'ordinateur.

C'est pour ces raisons qu'il nous a semblé intéressant de vous fournir quelques renseignements relatifs à l'écriture et à la lecture d'un enregistrement sur cassette.

Le gesticnaire du lecteur permet l'écriture ou la lecture d'un fichier sous un format prédéfini. Cette opération s'effectue caractère par caractère et est entièrement contrôlée par le logiciel.

L'écriture sur la cassette étant réalisée par le BIT 7 du port B du PPI, les données ne peuvent donc être lues ou enregistrées que bit par bit.

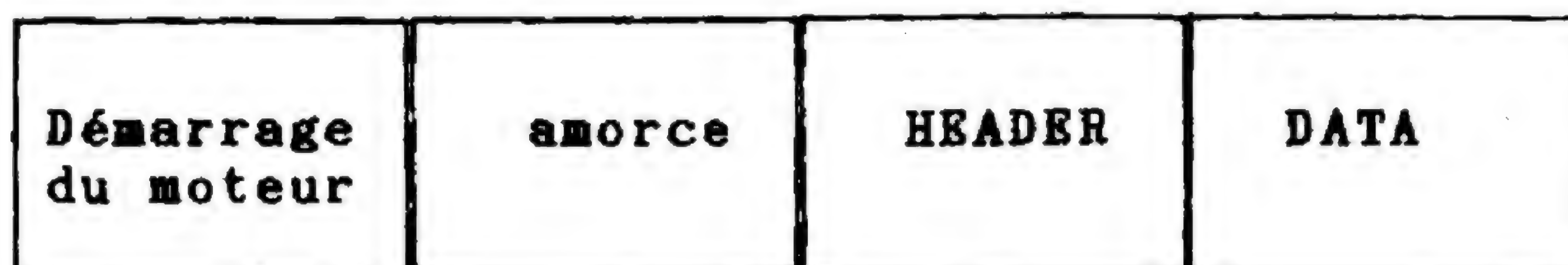
Chaque octet doit donc être décomposé en ses différents bits avant d'être transmis. Cette décomposition est réalisée par le logiciel.

Les bits sont transmis en commençant par le plus significatif (B7).

5.1.2 STRUCTURE GENERALE D'UN FICHIER.

Chaque fichier écrit sur la cassette peut comporter au maximum 65536 octets. Le fichier est décomposé en différents blocs commençant chacun par un HEADER. Ce HEADER est une tête de fichier, il est suivi d'un certain nombre d'octets (MAXIMUM 2048).

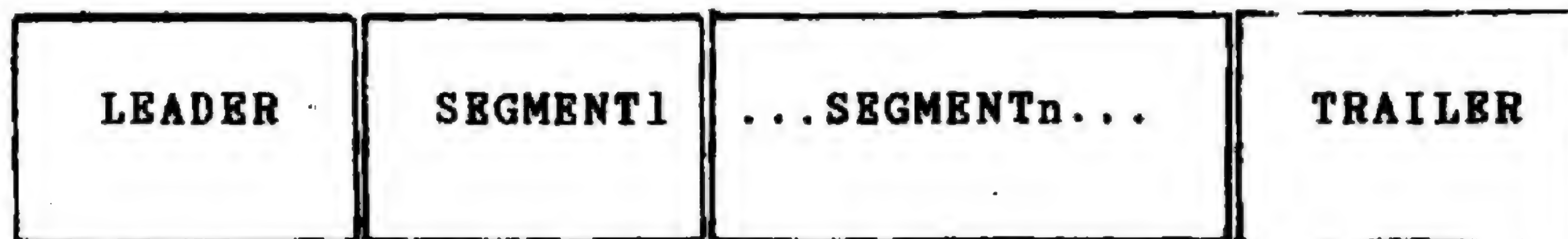
Lors de la commande d'écriture, après le démarrage du moteur du lecteur, une amorce est écrite sur la bande avant l'écriture du premier HEADER. Celle-ci permet lors de la lecture de bien séparer les différents blocs.



5.1.3 FORMAT D'ENREGISTREMENT.

Les données (DATA) sont découpées en blocs de 256 octets appelés SEGMENTS dont le dernier est complété par des zéros à droite lorsqu'il n'y a pas suffisamment d'octets pour le remplir. Lors de la lecture, aucun octet n'est ignoré.

FORMAT :



Un enregistrement est composé de 256 x N segments. Le nombre N détermine la longueur de l'enregistrement. En général N est compris entre 1 et 8.

A) LE LEADER

Au début de chaque enregistrement, un LEADER ou guide est écrit suivant le schéma :

intervalle de préenregistrement ou GAP	2048 bits = 1	bit = 0	octet de synchro
--	---------------	---------	---------------------

L'intervalle de préenregistrement ou GAP sert uniquement à empêcher le recouvrement de deux enregistrements consécutifs.

La longue séquence suivante est composée d'un segment dont tous les bits ont la valeur 1 (256 octets=2048 bits). Cette longue suite de 1 est nécessaire à l'ordinateur pour déterminer la vitesse (baud rate) à laquelle les données ont été écrites.

Le zéro suivant est utilisé pour marquer la fin du LEADER.

L'octet de synchronisation, en plus de sa fonction propre, sert également à distinguer l'enregistrement du HEADER de l'enregistrement des DATA. Le HEADER est écrit avec l'octet de synchronisation 2CH tandis que les données utilisent la valeur 16H. Cela permet à l'ordinateur, lors de la recherche d'un HEADER pour le gestionnaire, de ne pas le confondre avec une suite de données ou vice-versa.

B) LE SEGMENT.

Chaque segment contient 256 octets découpés selon le schéma suivant:

octet 1	octet 2	octet 256	CRC 1	CRC 2
---------	---------	-------	-----------	-------	-------

Le CRC 1 contient l'octet le plus significatif et le CRC 2, l'octet le moins significatif du résultat du calcul du CRC (Contrôle de redondance) pour les 256 octets du segment par le polynôme :

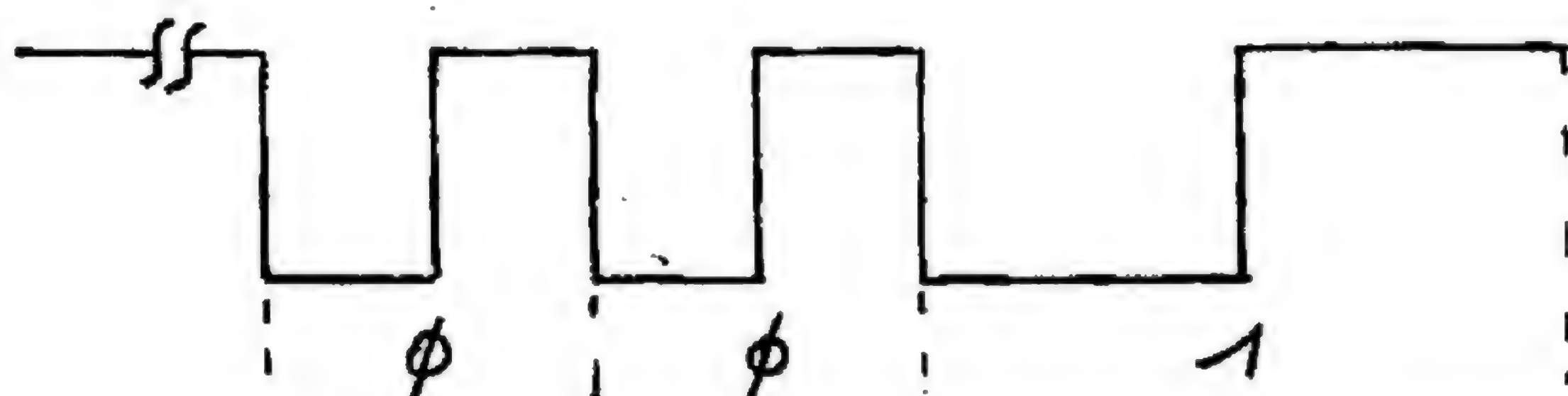
$$X^{15} + X^{12} + X^5 + 1 \quad (\text{valeur initiale 0FFFH})$$

C) LE TRAILER.

Le TRAILER est simplement une suite de 32 bits de valeur 1 qui termine l'enregistrement.

5.1.4 FORMAT D'ENREGISTREMENT DES BITS SUR LA CASSETTE.

Le signal fourni par le bit 7 du port B du PPI vers le lecteur de cassette est un signal carré. Chaque bit à transmettre est composé d'une phase zéro (low) suivi d'une phase 1 (high) de durées égales. On dit également que le signal carré présente un rapport cyclique de 1/1. Le temps nécessaire pour écrire un bit 0 vaut la moitié du temps nécessaire pour écrire un bit 1.



L'examen de cette technique nous permet de remarquer que les indications de vitesse d'écriture ne peuvent être qu'imprécises. En effet, elles dépendent essentiellement du nombre de 0 et de 1 écrits.

Par contre, l'étude statistique de la répartition des bits 1 et 0 sur un fichier nous montre que la probabilité de trouver un 1 ou un 0 est sensiblement identique. On peut dès lors s'en tenir aux indications de vitesse du manuel, 1000 bauds et 2000 bauds (1 baud = 1 bit par seconde), qui sont en réalité des valeurs moyennes.

5.1.5 LE HEADER

Le HEADER, ou en-tête, figure dans une zone de fichier longue de 64 octets et contient les informations relatives au fichier lui-même.

Il indique notamment si le fichier est protégé, s'il s'agit d'un fichier ASCII ou BASIC, la longueur du fichier, etc....

DESCRIPTION DES OCTETS.

octets 0 à 15: nom du fichier sur 16 octets ou rempli de 0.

octet 16 : numéro du bloc.

octet 17 : si cet octet contient la valeur 0, cela signifie qu'il s'agit du dernier bloc de l'enregistrement.

octet 18 : Cet octet contient le code du type de fichier:

BIT 0: le bit 0 est le bit de protection.
S'il vaut 1, le fichier correspondant est protégé.

BITS 1 à 3: déterminent le type de fichier:

B3	B2	B1	DECIMAL
----	----	----	---------

0	0	0	0	fichier BASIC
---	---	---	---	---------------

0	0	1	1	BINAIRE
---	---	---	---	---------

0	1	0	2	IMAGE
---	---	---	---	-------

0	1	1	3	ASCII
---	---	---	---	-------

Les autres valeurs ne sont pas utilisées.

BITS 4 à 7: ces bits comportent normalement des 0. Seuls les fichiers ASCII ont la valeur 1 dans le bit 4.

octets 19-20 : Ces octets contiennent la longueur des données du fichier. Si un bloc est entièrement écrit, les octets ont comme valeur 0800H, c'est à dire 2K (2048 octets).

octets 21-22 : Ils indiquent l'adresse de chargement à partir de laquelle les données ont été écrites à l'origine. Pour les programmes BASIC, cette adresse est égale à 0170H, soit 368 en décimal.

octet 23 : Si son contenu est différent de 0, il s'agit du premier bloc du fichier.

octets 24-25 : Contiennent la longueur totale du fichier.

octets 26-27 : Contiennent l'adresse d'exécution d'un programme binaire. Ils permettent de réaliser un AUTO-START par programmation.

Les octets 28 à 63 ne sont pas utilisés par le logiciel. Ils sont donc à la disposition de l'utilisateur.

5.1.6 VITESSE D'EXECUTION.

Le Basic Amstrad permet, grâce à l'instruction SPEED WRITE, de sélectionner la vitesse moyenne de stockage des données sur cassettes soit à 1000 bauds, soit à 2000 bauds. Ces deux possibilités ne sont pas limitatives. En effet, il existe une routine interne qui permet de sélectionner la vitesse d'écriture (CAS SET SPEED à l'adresse 0BC68H).

Cette routine requiert deux paramètres:

CE: Le registre HL du Z80 doit contenir la durée d'un demi-zéro en microsecondes. Le registre A ou accumulateur doit contenir la précompensation à appliquer.

CS: AF et HL sont modifiés, tous les autres registres sont préservés.

REMARQUES:

1- La vitesse d'exécution est définie par la durée d'un demi-zéro et la durée d'un bit 1 vaut le double de la durée d'un bit 0. La vitesse peut donc être relatée comme étant la moyenne des vitesses d'un bit 1 et d'un bit 0 en considérant que les nombres de bits à 0 et à 1 sont sensiblement identiques dans un fichier.

$$\begin{aligned}\text{baud rate (moyen)} &= 1\ 000\ 000 / (3 * \text{durée demi-zéro}) \\ &= 333\ 333 / (\text{durée demi-zéro})\end{aligned}$$

2- L'électronique du lecteur a tendance à faire varier la position des transmissions (zéro vers un ou un vers zéro). Pour palier à ce phénomène, on applique une compensation anticipée qui a pour but d'écrire plus brièvement les bits 0 et d'augmenter d'un même rapport la durée des bits 1. Pour 1000 bauds, les valeurs par défaut d'un demi-zéro et de la précompensation sont respectivement de 333 et de 25 micro secondes. Pour 2000 bauds, elles sont respectivement de 167 et 50 micro secondes. Ces valeurs ont été déterminées après de nombreux essais et l'utilisation d'autres valeurs ne sont pas garanties.

Pour tenter d'imposer une vitesse d'écriture plus rapide, par exemple 2400 bauds, il suffit de transmettre dans le registre HL la valeur:

$$\text{demi-zéro} = 333\ 333 / 2400 = 138,9 = 139 = 8BH$$

et dans l'accumulateur 60 ou 3CH.

Programme Basic:

```
10 MEMORY &6000
20 FOR I=&6001 TO &6009
30 READ A$
40 POKE I,VAL("&H"+A$)
50 NEXT I
60 DATA 21,8B,00,3E,3C,CD,68,BC,C9
70 CALL &6001
```

Programme assembleur contenu dans le programme Basic:

```
21 8B 00      LD HL,#8B
3E 3C         LD A,#3C
CD 68 BC      CALL #BC68
C9           RET
```

Ce programme n'est pas garanti. Il dépend non seulement du lecteur mais également de la qualité de la cassette utilisée. Bien entendu, vous pouvez, en faisant varier le contenu de HL et de A, trouver vous-même la vitesse moyenne de votre système.

5.1.7 ROUTINES DU GESTIONNAIRE CASSETTE

- #BC77 Positionne le tampon pour la lecture et lit le premier bloc.
CE: B contient la longueur du nom du fichier; HL contient l'adresse du nom du fichier et DE contient l'adresse du tampon (2 K).
CS: si Ok, le carry est vrai et le zéro est faux. HL contient l'adresse du tampon qui contient le HEADER, DE contient l'adresse des données, BC contient la longueur du fichier et A, le type de fichier.
Si le STREAM (flux) est déjà utilisé, le carry est faux et A, BC, DE et HL sont modifiés. Si on pousse sur ESC, le carry est faux et le zéro est vrai; AF, BC, DE et HL sont modifiés.
Dans tous les cas, IX est modifié.
- #BC7A Ferme le fichier.
CE: rien
CS: si Ok, le carry est vrai, sinon il est faux.
AF, BC, DE et HL sont modifiés dans les deux cas.

Citons encore:

- #BC65 initialisation du gestionnaire cassette.
- #BC68 positionnement de la vitesse d'écriture.
- #BC6B autorise ou interdit l'affichage des messages.
- #BC6E mise en route du moteur du lecteur.
- #BC71 arrêt du moteur du lecteur.
- #BC74 repositionne le moteur dans son état précédent.
- #BC7D abandonne la lecture et ferme le fichier.
- #BC80 lecture d'un octet.
- #BC83 lecture d'un fichier et écriture mémoire.
- #BC89 teste si on atteint le fin de fichier.
- #BC8C ouverture d'un fichier en sortie.
- #BC8F fermeture d'un fichier en sortie.
- #BC92 fermeture immédiate d'un fichier en sortie.
- #BC95 écriture d'un caractère sur un fichier de sortie.
- #BC98 écriture directe du contenu d'une mémoire vers un
fichier de sortie.
- #BC9B génère le catalogue cassette.
- #BC9E écrit un enregistrement sur cassette.
- #BCA1 lit un enregistrement sur cassette.
- #BCA4 compare un enregistrement sur cassette avec le
contenu de la mémoire.

5.2 L'interface imprimante Centronics.

L'interface imprimante est certainement la partie hardware la plus simple de l'Amstrad.

Il est réalisé au moyen d'un circuit 74 LS 173 composé de 8 latches travaillant comme des flip-flops (bascules) et est interfacé à l'adresse EFXH.

Les entrées du latch sont directement reliées au bus du microprocesseur. Les sorties, quant à elles, sont connectées au bus data de l'imprimante. Seul le bit B7 est envoyé au port centronics à travers un inverseur pour générer le strobe, c'est à dire le signal de commande d'impression d'un caractère pour l'imprimante. Il est normalement à 1.

Pour envoyer un caractère, l'ordinateur doit premièrement introduire dans le latch l'octet à imprimer et positionner peu après le signal strobe à 0 de la manière suivante:

NOTE: le caractère à imprimer étant codé en ASCII, seuls les bits B6 à B0 sont utilisés. Le bit B7 est utilisé comme strobe.

On opère sur chaque octet à envoyer une instruction AND avec la valeur hexadécimale 7FH

octet à envoyer =	xxxx xxxx
7F =	0111 1111

AND =	0xxx xxxx

Après cette opération, nous sommes certains que le bit B7 est bien à 0 et le strobe du à l'inverseur, à 1.

Cet octet est transmis à l'imprimante par l'instruction

OUT &HEF00,&X0xxxxxxx

Pour être imprimé, le signal strobe doit passer à l'état bas. Cette opération est effectuée en envoyant sur le port de sortie le même octet préalablement traité par l'instruction OR 80H:


```

octet à envoyer = 0xxx xxxx
                   80 = 1000 0000
                   -----
                   OR = 1xxx xxxx

```

Le caractère imprimé, le strobe doit de nouveau revenir à son état initial qui est l'état 1. C'est pourquoi l'octet est encore une fois envoyé, mais le bit B7 est positionné à l'état 0 par l'opération AND 7F.

Il est à remarquer que le caractère est imprimé seulement si le signal BUSY de l'imprimante est à l'état bas. Nous rappelons que le signal BUSY est interrogé par le bit B6 du port B du PPI.

Dans le cas contraire, l'octet transmis à l'imprimante peut être perdu. De plus, l'octet à imprimer doit se trouver au moins une microseconde avant le début du signal strobe et une microseconde après la fin de celui-ci. La durée du strobe doit être comprise entre 1 et 500 microsecondes.

Voici maintenant un exemple de programme qui a pour but d'imprimer des lignes de B:

1) programme Basic:

```

10 MEMORY &H6000
20 FOR I=&H6001 TO &H6011
30 READ A$
40 POKE I,VAL("&H"+A$)
50 NEXT I
60 DATA 06,EF,3E,42,E6,7E,ED,79,F6,80,ED,79,E6,7E,
      ED,79,C9
70 CALL &6001
80 GOTO 70

```

2) programme en assembleur contenu dans le programme Basic:

06 EF	LD B,#EF
3E 42	LD A,#42
E6 7E	AND #7E
ED 79	OUT (C),A
F6 80	OR #80
ED 79	OUT (C),A
E6 7E	AND #7E
ED 79	OUT (C),A
C9	RET

La sortie sur imprimante ne pose pas de problème en Basic (voir manuel).

En assembleur, bien que la procédure ne soit pas très compliquée, quand un grand nombre de caractères doit être imprimé, la gestion devient plus lourde (test de l'imprimante, des sauts de ligne (LF), des retours de ligne (CR), etc...).

Le système d'exploitation possède donc des routines internes qui vous évitent une grosse part du travail d'écriture de la procédure:

- #BD2B Cette routine envoie un caractère à l'imprimante, avec la possibilité de retour si l'imprimante est occupée.
CE: A contient le caractère à envoyer.
CS: si le caractère a été envoyé, le carry est vrai. Si l'imprimante reste occupée trop longtemps, le carry est faux. De toutes façons, AF est modifié.
- #BD2E Teste si l'imprimante est occupée. Si oui, le carry est vrai, sinon il est faux.
- #BD31 Envoie un caractère à l'imprimante (elle ne doit pas être occupée).
CE: A contient le caractère à envoyer.
CS: le carry est vrai si le caractère est envoyé et AF est modifié.

L'exemple suivant produit le même résultat que l'exemple précédent mais en utilisant la routine #BD31:

1) programme Basic:

```
10 MEMORY &H6000
20 FOR I=&H6001 TO &H6006
30 READ A$
40 POKE I,VAL("&H"+A$)
50 NEXT I
60 DATA 3E,42,CD,31,BD,C9
70 CALL &6001
80 GOTO 70
```

2) programme en assembleur contenu dans le programme Basic:

```
3E 42          LD A,#42
CD 31 BD       CALL #BD31
C9            RET
```


5.3 Le clavier.

5.3.1 FONCTIONNEMENT.

Nous avons vu à la section 4.5 (page 87 à 90) la description matérielle du clavier. L'étude complète des routines de lecture primaire à travers le PPI réalisée dans cette section ne permet pas une compréhension totale du fonctionnement de ce périphérique. C'est pourquoi nous allons développer à présent le fonctionnement interne du logiciel clavier.

La matrice clavier étudiée au chapitre 4 est donc composée de 10 lignes. Ces lignes sont scrutées par le logiciel tous les cinquantièmes de seconde (interruption). La lecture des 10 lignes composées de 8 colonnes fournit 10 octets. Ces 10 octets sont mémorisés aux adresses B4EB à B4F4 pour le CPC464 (B635 à B63F pour le CPC664).

Les 10 octets sont traités pour fournir un numéro de touche correspondant à la touche enfoncée.

A l'issue de cette lecture, Les états des touches SHIFT et CONTROL sont analysés pour déterminer le type de caractère tapé. La touche ESC (BREAK) est également analysée à cet instant.

Les états des touches SHIFT et CONTROL fournissent 4 possibilités :

- 1 ni SHIFT ni CONTROL ne sont enfoncés.
- 2 SHIFT est enfoncé et CONTROL ne l'est pas.
- 3 SHIFT n'est pas enfoncé et CONTROL l'est.
- 4 SHIFT et CONTROL sont enfoncés.

Pour des raisons d'uniformité des claviers, les possibilités 3 et 4 sont confondues. Autrement dit, si la touche CONTROL est enfoncée, l'état de la touche SHIFT n'a aucune importance.

Pour les amateurs, rappelons que la touche SHIFT positionne dans la plupart des cas le BIT 6 du code ASCII à 1 et que la touche CONTROL positionne toujours le BIT 7 du code ASCII à 0.

Le système possède 3 tables, une pour chacune des possibilités décrites ci-dessus. Grâce à la valeur de la touche calculée précédemment, une valeur est extraite de la table choisie. Cette valeur peut être un code ASCII ou un code spécial. Le code spécial indique qu'un traitement particulier doit être envisagé.

00H à 7FH identifient des caractères ASCII standard.

80H à 9FH identifient des codes de compression qui doivent être traduits par une table supplémentaire (touches programmables).

A0H à FCH identifient des caractères graphiques.

FDH est réservé au CAPS LOCK (verrouillage majuscule)

FEH est réservé au SHIFT

FFH signifie que la touche enfoncée doit être ignorée.

Enfin une table supplémentaire est consultée pour déterminer si la touche est dotée d'une faculté de répétition.

Adresses des tables.

Table	CPC 464	CPC 664
ni SHIFT ni CONTROL	B34C	B496
SHIFT et pas CONTROL	B39C	B4E6
CONTROL	B3EC	B536
Faculté de répétition	B43C	B586
Table des valeurs des codes étendus.	B445	B591

Une fois le caractère ou la suite de caractères identifié, le résultat est introduit dans un tampon (BUFFER) réservé au clavier. Ce tampon sera 'vidé' par les gestionnaires connexes.

La gestion de la touche ESC est particulière. Un premier appui doit 'geler' l'écran et un second appui doit produire un 'BREAK'. En outre si SHIFT et CONTROL sont enfoncés, c'est un 'RESET' qui doit se produire.

La possibilité d'interdire le BREAK ou de le dérouter par logiciel (ON BREAK GOTO...) implique une gestion complexe de cet événement. Une multitude de routines et de variables systèmes sont donc nécessaires pour une saine gestion de la touche ESC.

L'appui sur la touche ESC produit l'introduction d'un caractère marqueur spécial dans le tampon clavier et l'activation d'un événement BREAK dans le gestionnaire d'événement. Le caractère enfoncé par la suite déterminera l'action à prendre (ignorance, interruption du programme...).

En plus des touches du clavier, le gestionnaire s'occupe de la gestion des manettes de jeux. Comme nous l'avons vu au chapitre 4, les joysticks sont interfacés dans la matrice clavier et il n'y a rien de spécial à dire à leur sujet.



5.3.2 Les routines du gestionnaire clavier

BB00	Initialisation
BB03	RESET
BB06	Attente de caractère clavier (caractère dans A).
BB09	Lecture du clavier sans attente.
BB0C	Réserve un caractère pour le prochain appel de BB09.
BB0F	Positionne une chaîne de caractères associée à un code étendu.
BB12	Lit un caractère depuis une chaîne expansée.
BB15	Allocation d'un tampon pour une chaîne expansée.
BB18	Attente de caractère clavier (dans A)
BB1B	Teste disponibilité d'une touche.
BB21	Etat du CAPS LOCK
BB24	Lecture JOYSTICK (dans A, H et L)
BB27	Positionne un code dans table sans SHIFT ni CONTROL.
BB2A	Fournit le code de la touche pressée sans SHIFT ni CONTROL.
BB2D	Positionne un code dans table SHIFT.
BB30	Fournit le code de la touche pressée avec SHIFT.
BB33	Positionne un code dans table CONTROL.
BB36	Fournit le code de la touche pressée avec CONTROL.
BB39	Positionne un code dans la table de répétition.
BB3C	Teste si la touche enfoncée doit se répéter.
BB3F	Positionne le temps avant le début de répétition et la vitesse de répétition.
BB42	Lecture du temps avant répétition et de la vitesse de répétition.
BB45	Arme le mécanisme de BREAK.
BB48	Désarme le mécanisme de BREAK.
BB4B	Génère un BREAK.

Vous trouverez tous les détails et les conditions d'entrée et de sortie dans CLEFS POUR AMSTRAD.

STRUCTURE INTERNE DU BASIC AMSOFT.

6.1 Généralités.

Les deux ROMs AMSOFT contiennent un système d'exploitation rudimentaire, un interpréteur BASIC, un ensemble de routines mathématiques, un ensemble de routines de gestion des périphériques et une copie du générateur de caractères.

Le propos de ce chapitre est de décrire les opérations fondamentales des ROMs pour que vous puissiez en tirer le meilleur parti lors de la programmation en assembleur.

Un ordinateur sans système d'exploitation présente peu d'intérêt. Le système d'exploitation permet la communication entre l'utilisateur et la machine, ce qui signifie : lire le clavier pour "voir" si on appuie sur une touche et écrire les messages à l'écran.

Lorsque nous écrivons un programme, il y a un programme dans l'ordinateur qui reçoit nos ordres d'écriture. C'est le système d'exploitation.

6.2 Composition des ROMs.

Les ROMs se composent de :

- Un certain nombre de DRIVERS (programmes d'interfaçage) pour chacun des périphériques comme le clavier, la cassette et l'imprimante.
- Des routines mathématiques et arithmétiques.
- Des routines de gestion de la mémoire et des tables.
- Un moniteur, programme qui consulte continuellement le clavier en attente d'une entrée.
- Divers utilitaires comme l'éditeur , le gestionnaire d'interruptions...
- Un ensemble de tables (générateur de caractères, conversion du clavier, codes de contrôle de l'écran).
- L'interpréteur syntaxique du langage BASIC

Le dernier module (l'interpréteur) se trouve dans la ROM supérieure, les autres se trouvent dans la ROM inférieure.

6.3 Découpage de la mémoire centrale.

La mémoire centrale est découpée de la façon suivante :

0000	ZONE COPIE DE LA ROM INFÉRIEURE
0170	ZONE DES PROGRAMMES BASIC
XXXX	ZONE DES VARIABLES
YYYY	ZONE DE LA PILE
AXXX	ZONE DE COMMUNICATION
0000	MEMOIRE ECRAN
FFFF	FIN DE LA MEMOIRE

La zone pour les programmes et les variables peut être divisée en deux tables principales.

A - La TIP : Table des Instructions du Programme.

B - La TV : Table des Variables.

Insérer ou effacer une ligne BASIC d'un programme produit un accroissement ou une réduction de la TIP, de même façon, définir une nouvelle variable accroît la taille de la TV. Comme les adresses de début des tables sont variables, elles sont définies à un endroit fixe de la région de communication. ce principe permet de déplacer les tables où l'on veut et de toujours savoir où elles sont situées.

La TV contient le nom et la valeur de chaque variable contenue dans le programme. Elle est divisée en 2 sous-tables définies en fonction du type de variable.

6.4 Structure de la Table des Instructions de Programme.

La TIP contient les lignes de programme BASIC, elle commence en standard à l'adresse 170H (368).

Toutes les lignes d'un programme ont la même structure. On trouve d'abord deux octets qui indiquent la longueur de la ligne en hexadécimal. Cette longueur comprend tous les octets de la ligne y compris le 0 final et les deux premiers octets. En ajoutant cette valeur à l'adresse courante, on trouve l'adresse du début de la ligne suivante. Ensuite, on trouve le numéro de ligne courante en binaire sur deux octets, ensuite, on trouve le contenu de la ligne avec les mots clés sous forme de codes, et enfin, un octet égal à 0 pour indiquer la fin de la ligne.

A la fin du programme, après la dernière ligne, l'adresse de la ligne suivante est remplacée par 2 octets qui valent 00. Un programme BASIC se termine donc toujours par 3 octets 00 (1 de fin de ligne + 2 de fin de programme).

Exemple : le programme : 10 PRINT "MARTIN"
 suivi de : 20 A\$="JADOUL"
est mémorisé en hexadécimal dans la TIP de la façon suivante:

Adr.	val.	explication	:	Adr.	val.	explication
170	0E	longueur ligne 10:		181	00	ligne 20
171	00	= 14 octets	:	182	03	identifieurs
172	0A	numéro de ligne	:	183	05	de la variable
173	00	en hexa 2 octets	:	184	00	A\$ type alpha
174	EF	code du PRINT	:	185	C1	code du A + 80H
175	22	"	:	185	EF	code de =
176	4D	M	:	187	22	"
177	41	A	:	183	4A	J
178	52	R	:	183	41	A
179	54	T	:	18A	44	D
17A	49	I	:	183	4F	O
17B	4E	N	:	18C	55	U
17C	22	"	:	18D	4C	L
17D	00	fin de ligne	:	18E	22	"
17E	12	longueur ligne 20:		18F	00	fin de ligne
17F	00	18 octets	:	190	00	fin de programme
180	14	numéro de ligne	:	191	00	fin de programme

6.5 Structure de la Table des Variables (TV).

Cette table contient toutes les variables définies dans le programme BASIC. Elle est divisée de façon interne en deux parties, la première contient des informations sur toutes les variables simples (non dimensionnées) et la deuxième contient des informations sur toutes les variables dimensionnées.

Comme la TIP, la TV se trouve en RAM et il y a 1 pointeur pour chacune des deux parties dans la région de communication. L'adresse AE87H (CPC464) contient l'adresse de la table des variables simples, et l'adresse AE89H (CPC464) contient l'adresse de la table des variables dimensionnées.

Remarque : les deux adresses ci-dessus sont valables pour le CPC464 uniquement. Pour le CPC664 reportez vous au chapitre 10.

La première partie de la TV commence en général à la fin de la TIP. La seconde partie commence à la fin de la première.

La structure de la table est fonction du type de variable. Pour plus de détails sur cette structure, reportez vous à l'étude de la fonction @ (VARPTR) au chapitre 7.

Les variables de type chaîne (alphanumérique) sont représentées dans la TV par un pointeur. Le contenu réel d'une variable alphanumérique se trouve dans une autre table appelée Table des Chaînes.

Les variables sont disposées dans la TV au fur et à mesure de leur apparition dans le programme. Il n'y a pas d'ordre alphabétique. Les variables dimensionnées sont souvent déplacées, car l'apparition d'une nouvelle variable non dimensionnée produit un déplacement total de la zone des variables dimensionnées.

Exemple :

```
10 DIM A(5)
20 PRINT @A(0)
30 B=@A(0)
40 PRINT B
```

Cet exemple produira deux valeurs différentes pour l'adresse de la variable A(0), car entre la ligne 20 et la ligne 40, une nouvelle variable simple (B) est apparue.

Remarque : Pour des explications sur la fonction @ utilisée dans l'exemple ci-dessus, reportez-vous au chapitre 7.

Les variables multidimensionnées sont mémorisées dans l'ordre des colonnes. De cette façon, un déplacement de l'index gauche est plus rapide qu'un déplacement de l'index droit.

De cette structure découle une règle simple de conception de programmes optimisés :

Déclarez toutes vos variables simples en début de programme. De cette façon, la première partie de la table sera fixe et de nombreux mouvements des variables dimensionnées seront évités.

5.6 Structure de la Table des Chaînes (TC).

La table des chaînes se trouve en haut de mémoire. Elle fonctionne dans le sens inverse de la table des variables. Autrement dit, un pointeur de la région de communication pointe sur la table des chaînes et à chaque apparition d'une nouvelle chaîne, le contenu de ce pointeur est décrémenté.

Chaque élément de la table est composé de deux octets indiquant la longueur de la chaîne suivis de la chaîne elle-même.

Exemple :

```
10 A$="BONJOUR"  
20 B$="HELLO"
```

Si x est l'adresse supérieure de l'espace disponible, alors ce programme produira :

x-15	x-12	x-10	x-8	x-6	x-4	x-2	x								
05	00	H	E	L	L	O	07	0C	I	O	N	J	O	U	R

----- Longueur de BONJOUR (7).

----- Longueur de HELLO codé sur 2 octets (5 caractères).

6.7 La région de communication.

La région de communication est la zone supérieure de la mémoire RAM située juste en dessous de la mémoire écran (la région de communication se termine en BFFFH).

Elle se divise en deux grandes parties.

1) La zone de mémorisation des paramètres et des variables internes. Cette zone contient principalement des éléments de 1 octet (variables internes et sémaphores) et des éléments de deux octets (variables internes et adresses de tables ou de routines). Dans le CPC464 cette zone est comprise entre les adresses AC00 et B8F7.

Une table des principales variables internes se trouve dans le livre CLEFS POUR AMSTRAD.

2) La table des vecteurs qui permet l'appel des routines internes contenue dans les ROMs ainsi que l'interception de certaines routines (INDIRECTION) pour modifier leur effet. Cette table commence en B900 et se termine en BFFF.

6.8 Fonctionnement du BASIC.

Le déroulement du BASIC se fait en deux phases.

1° phase : la phase d'entrée.

Elle accepte les entrées en provenance du clavier (rédaction de programmes). Après l'impression du message 'Ready', le système est en phase d'entrée.

Fonctionnement succinct de la phase d'entrée :

- A - Lire la ligne entrée au clavier.
- B - Remplacer les mots clés par leurs codes (TOKEN).
- C - Tester si c'est une instruction directe (RUN, LOAD, absence de numéro de ligne).
- D - Dans la négative, mémoriser dans la TIP.

2° Phase : la phase d'interprétation et d'exécution.

Le BASIC est un interpréteur, les lignes sont donc analysées et exécutées les unes après les autres. Quand on exécute le programme (RUN), le système cherche un code de mot réservé. Une fois ce code trouvé, une adresse est associée à ce code et le contrôle est passé à cette adresse.

Ces différentes adresses sont les points d'entrées des routines de traitement des instructions.

La routine appelée teste la syntaxe de l'instruction (position des virgules, des parenthèses,...).

La phase d'exécution démarre avec une instruction RUN ou GOTO ou lorsqu'une instruction sans numéro de ligne est entrée.

Fonctionnement succinct de la phase d'exécution.

- A - Prendre le premier caractère de la ligne courante (TIP).
- B - Si le caractère n'est pas un code de mot clé, sauter à F
- C - Rechercher l'adresse de la routine correspondante.
- D - Exécuter la routine en question.
- E - Retourner à A.
- F - Assigner la variable.
- G - Evaluer l'expression qui suit la variable.
- H - retourner à A.

La routine d'exécution commence par charger le premier caractère de la ligne courante. Ce caractère est testé; s'il est supérieur à 80H (128) c'est un code représentant une instruction, le contrôle est alors passé à la routine associée à cette instruction; s'il est inférieur à 80H, c'est une affectation de variable de la forme X=fonction. La routine d'analyse d'affectation prend le nom de la variable, teste si elle est suivie d'un signe = puis évalue l'expression qui suit le signe =.

Si un code a été trouvé, la routine analyse si ce code est correct car certains codes ne peuvent pas apparaître seuls (THEN, ELSE ...) et aucune des fonctions du BASIC ne peut apparaître à gauche du signe = .

Enfin le code est analysé et le contrôle est donné à la routine associée à ce code.

Après chaque routine d'interprétation, un test est effectué pour déterminer s'il y a une marque de fin de ligne ou un symbole ':' de ligne multi-instructions.

En fin de programme, le contrôle est automatiquement donné à la routine de traitement de l'instruction END, même si celle-ci est absente.

6.9 Fonctions arithmétiques et mathématiques de la ROM.

Avant d'analyser les fonctions de la ROM, rappelons les fonctions intrinsèques du processeur Z80.

Le Z80 est capable de réaliser des additions et des soustractions d'entiers de 8 ou de 16 bits. Il ne permet pas la multiplication ou la division.

Ces opérations sont permises entre les registres. Le Z80 ne possède pas d'instructions de calcul entre la mémoire et les registres.

Le BASIC, par contre, supporte les quatre opérations fondamentales avec des variables en virgule flottante.

Ces opérations sont réalisées grâce à des routines internes de la ROM.

A cause de la complexité de ces routines et du type de variables utilisées, les registres internes du processeur ne suffisent pas. Une zone tampon dans la zone de communication doit être utilisée. Cette zone est dénommée ACCUMULATEUR VIRTUEL.

L'accumulateur virtuel est une zone de six octets situés dans le CPC464 de l'adresse B0C1 à l'adresse B0C6.

L'octet B0C1 est appelé sémaphore de type de variable. Il contient un nombre représentant le type de la variable (2=entier, 3=chaîne, 5=réel).

Le codage de l'accumulateur virtuel dépend du type de la variable.

- Une variable entière est codée sur 2 octets en binaire signé.
- Une variable réelle est codée sur 5 octets. Pour les explications sur le codage des variables réelles, reportez-vous au chapitre 7.

6.10 La PILE.

Il existe un espace réservé pour les adresses de retour des sous-routines, les boucles FOR-NEXT et les WHILE-WEND. C'est l'espace réservé à la pile d'adresse.

Cette pile n'est pas celle du processeur, mais une zone mémoire pointée par un vecteur RAM situé dans le CPC464 à l'adresse B08B et B08C. Ce vecteur pointe une zone située entre AE8B et B08A.

A chaque rencontre d'une instruction GOSUB, FOR ou WHILE, l'interpréteur BASIC pousse un certain nombre d'octets dans la pile afin de pouvoir les dépiler ensuite.

Avant chaque nouvelle allocation d'espace, la routine de gestion de mémoire effectue un test pour déterminer s'il reste un espace mémoire suffisant (si la PILE ne va pas dépasser l'adresse B08A). Si la place est insuffisante, le système produit le message d'erreur : MEMORY FULL.

Toutes les variables associées à une boucle FOR sont transmises dans la pile jusqu'à la fin de la boucle. Quand l'interpréteur rencontre une instruction NEXT, il effectue une recherche dans la pile pour retrouver une FRAME (suite d'octets) qui porte sur la même variable index. Si cette FRAME n'est pas trouvée, l'interpréteur produit le message Unexpected NEXT.

Format de la FRAME de l'instruction FOR

La FRAME FOR utilise 16 octets si la variable de boucle est entière et 22 octets si la variable est réelle.

Bas de mémoire :	Adresse de la variable index :	2 octets
	: Valeur qui suit le TO :	2/5 octets
	: Valeur du pas (STEP) :	2/5 octets
	: Signe de l'incrément + ou - :	1 octet
	: Adresse de l'instruction FOR :	2 octets
	: Adresse de la ligne du FOR :	2 octets
	: Adresse de l'instruction NEXT :	2 octets
	: Adresse de la ligne du NEXT :	2 octets
	: constante 10H ou 16H (taille) :	1 octets

Lors de l'apparition d'une instruction GOSUB, l'interpréteur pousse une FRAME dans la pile, et lors de l'apparition d'une instruction RETURN, il fouille la pile pour retrouver la FRAME du GOSUB la plus proche. S'il n'y a pas de FRAME, il produit le message 'Unexpected RETURN'.

Format de la FRAME GOSUB

La FRAME GOSUB utilise 6 octets.

Adresse basse :	Type de GOSUB 0=normal	:	1 octet
	1=GOSUB AFTER ou EVERY		
	: Adresse de l'instruction qui	:	2 octets
	suit le GOSUB		
	: Adresse de la ligne du GOSUB	:	2 octets
	: Constante 6 (taille FRAME)	:	1 octet

Format de l'instruction WHILE WEND

Le format de cette instruction est similaire à celui du GOSUB, mais il occupe 7 octets.

Adresse basse :	Adresse de la ligne du WHILE	:	2 octets
	: Adresse de l'instruction WEND	:	2 octets
	: Adresse de l'instruction WHILE	:	2 octets
	: Constante 7 (taille FRAME)	:	1 octet

6.11 Décomposition des ROMs.

Les adresses données ici sont valables pour le CPC 464 uniquement. Pour le CPC 664, reportez-vous au chapitre 9.

A - La ROM inférieure :

Adr. à adr. - - - - - C O N T E N U - - - - -

0000 à 003B Points d'entrée des RESTARTS.

0000	Initialisation complète (à l'allumage).
0008	Saut ROM ou RAM inférieure (adresse dans SP).
000B	Saut ROM ou RAM inférieure (adresse dans HL).
000E	Saut à l'adresse contenue dans BC.
0010	Appel d'une ROM extérieure (adresse dans SP).
0013	Appel d'une ROM extérieure (adresse dans HL).
0016	Saut à l'adresse contenue dans DE.
0018	Appel d'une sous-routine en ROM ou RAM (adresse SP).
001B	Appel d'une sous-routine en ROM ou RAM (ad. HL + C).
001E	Saut à l'adresse contenue dans HL.
0020	Charge dans A le contenu de HL (toujours en RAM).
0023	Appel sous-routine ROM ou RAM (HL pointe l'adresse).
0028	Saut dans la ROM inférieure (adresse dans SP).
0030	Restart réservé pour l'utilisateur.
0038	Point d'entrée des interruptions matérielles.
003B	point d'entrée des interruptions externes.

005C à 029B Routines de traitement des interruptions et des queues de blocs d'évènements.

02A1	Introduit un RSX dans le système.
02B2	Recherche un RSX pour exécuter une commande.
0329	Initialisation des ROMS externes.
0332	Initialise une ROM externe particulière.
05DC	Charge et exécute un programme.
060B	Lance un programme externe.
066D	Message : 64K MICROCOMPUTER (V1)
068A	Message de copyright.
06F4	Message : program load failed.
0727	Liste des noms des compatibles.

0776 Positionne le mode écran.
 0786 Positionne toutes les encres dans une seule couleur.
 0799 Positionne les couleurs des encres et du bord.
 07BA Attend le retour de balayage.
 07C6 Positionne l'OFFSET écran.

 07E6 à 0825 Manipulation du port imprimante.

 0826 Envoi de données au PSG.

 0AA0 à 1077 Routines de gestion du 'SCREEN PACK'
 Gestionnaire d'écran primaire.

 1078 à 15AF Routines de gestion du 'TEXT PACK'
 Gestionnaire d'écran en mode caractère.

 146B Table des codes de contrôle du terminal (96 octets).

 15B0 à 19DF Routines de gestion du 'GRAPH PACK'
 Gestionnaire d'écran en mode graphique.

 19E0 à 1E67 Routines de gestion du clavier.

 1AB3 Table des valeurs par défaut des touches étendues.
 1D69 Table des valeurs par défaut des touches normales.

 1E68 à 236F Routines de gestion du PSG.

 2370 à 2E17 Routines de gestion de la cassette.

 27C5 Message : press play then any key.
 27DB Message : error.
 27E5 Message : REC.
 27E8 Message : and.
 27ED Message : Read.
 27F3 Message : write.
 27FA Message : Rewind.
 2800 Message : tape.
 2805 Message : found.
 280D Message : loading.
 2815 Message : saving.
 281D Message : OK
 2820 Message : Block.
 2826 Message : Unnamed.
 282D Message : File.

 2E18 à 37FF Routines du 'MATH PACK'. (Pour une description
 complète des routines: voir CLEFS POUR AMSTRAD).

 2F53 Table de 65 octets contenant les puissances de 10.
 3086 Valeur de LOG(2).

308C	valeur de LOG10(2).
30CC	valeur 0.5
30FB	Constante 1,44269504
3100	Constante 88,0296919
3105	Constante -88,7228391
31A9	Constante PI 3,14159265
31EC	Table des constantes pour les sinus (30 octets).
321D	Idem (20 octets).
3258	Table des constantes pour ARC TANGENTE (55 octets).

3800 à 3FFF Table du générateur de caractères.

B - La ROM supérieure :

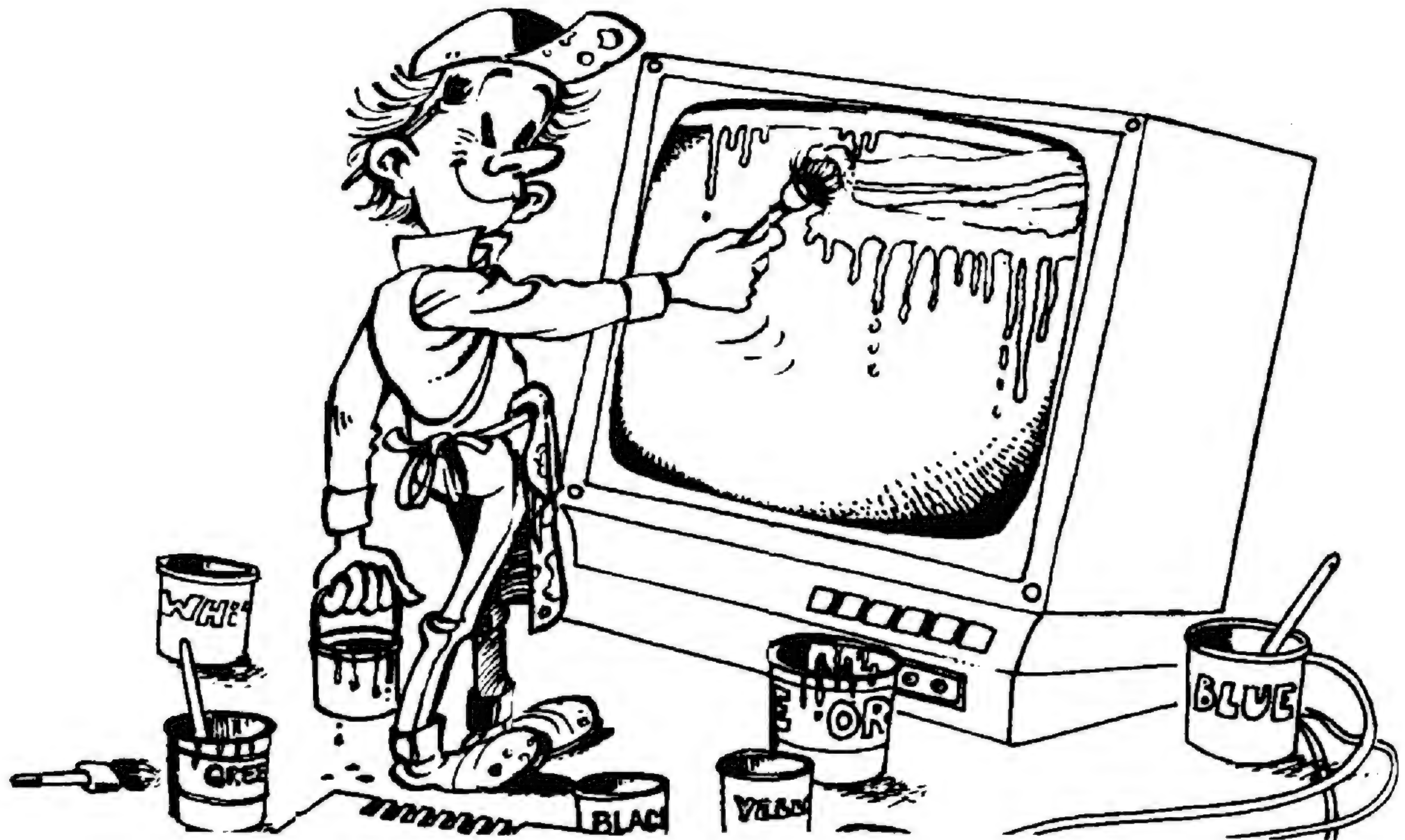
C002	Initialisation + envoi du message BASIC 1.0
C03F	message BASIC 1.0
C053	fonction EDIT
C090	entrée principale (affichage du READY)
C0CC	message READY
C0DF	AUTO
C12B	NEW
C132	CLEAR
C20A	PAPER
C212	PEN
C221	BORDER
C22A	INK
C24F	MODE
C25A	CLS
C262	VPOS
C276	POS
C2D2	LOCATE
C2E1	WINDOW
C319	TAG
C320	TAGOFF
C337	envoie le message pointé par HL
C3E3	WIDTH
C417	EOF
C48C	ORIGIN
C4B5	CLG
C4C6	DRAW
C4CB	DRAWR
C4D0	PLOT
C4D5	PLOTR
C4E9	TEST
C4EB	TESTR
C505	MOVE
C50A	MOVER
C529	FOR

C5FB	NEXT
C6C7	IF
C6E8	GOTO
C6ED	GOSUB
C70F	RETURN
C747	WHILE
C776	WEND
C7C3	ON
C8CB	ON BREAK
C8E1	DI
C8E7	EI
C940	ON SQ
C971	AFTER
C979	EVERY
C99F	REMAIN
CA8F	ERROR
CB23	message UNDEFINED LINE
CB33	routine envoi message BREAK in
CB4F	message BREAK
CB55	message IN
CB5A	STOP
CB65	END
CBC0	CONT
CBF8	ON ERROR
CC03	RESUME
CC5B	table des messages d'erreur
CE66	fin de la table des messages d'erreur
CF81	table des points d'entrée des opérations arithmétiques et logiques.
DOCA	table des points d'entrée des fonctions EOF, ERR, HIMEM, INKEY\$, PI, RND, TIME, XPOS et YPCS
D0DC	ERR
D0F4	HIMEM
D107	XPOS
D10E	YPOS
D190	table des points d'entrée des fonctions
D219	ROUND
D1EA	MIN
D1EE	MAX
D256	OPENOUT
D25F	OPENIN
D298	CLOSEIN
D2A1	CLOSEOUT
D2C0	SOUND
D31E	RELEASE
D329	SQ
D34E	ENV
D385	ENT
D409	INKEY
D423	JOY
D439	KEY DEF

D494	SPEED
D4B	PI
D4E7	DEG
D4EB	RAD
D4EF	SQR
D4F4	routine d'élévation à une puissance.
D520	EXP
D525	LOG10
D52A	LOG
D52F	SIN
D534	COS
D539	TAN
D53E	ATN
D543	message RANDOM NUMBER SEED ?
D559	RANDOMIZE
D584	RND
D614	DEFSTR
D618	DEFINT
D61C	DEFREAL
D654	LET
D67D	DIM
D9C0	ERASE
DAF8	LINE
DB28	INPUT
DB77	message ?redo from start
DCD9	RESTORE
DCEB	READ
DDEZ	TRON
DDEG	TROFF
DE01	table des points d'entrée des mots clé BASIC
DEBA	fin de la table
DFDC	table des mots clé qui peuvent être suivis d'un numéro de ligne (GOTO, RESTORE, AUTO, EDIT,...)
E0F7	LIST
E2DD	routine de positionnement sur la table des lettres pour rechercher un mot clé
E327	routine de test qui vérifie si un mot clé se trouve dans la table
E354	table des adresses pour chacune des 26 lettres de l'alphabet
E383	table des mots clé avec leur code
E64A	fin de la table
E723	DELETE
E7DF	RENUM
E8EF	DATA
E8F3	REM
E9BD	RUN
E9F6	LOAD
EA3C	CHAIN
EAA6	MERGE
EC09	SAVE

F158	PEEK
F15F	POKE
F16D	INP
F177	OUT
F17D	WAIT
F1BA	CALL
F1F6	ZONE
F1FD	PRINT
F2C4	PRINT USING
F47B	WRITE
F4EF	MEMORY
F69D	SYMBOL
F834	LOWER\$
F839	routine de conversion en minuscules
F842	UPPER\$
F8BA	BIN\$
F8C4	HEX\$
F8EA	DEC\$
F91E	STR\$
F93C	LEFT\$
F943	RIGHT\$
F993	MID\$
FA0A	LEN
FA10	ASC
FA16	CHR\$
FA24	INKEY\$
FA36	STRING\$
FA57	SPACE\$
FA77	VAL
FAA1	INSTR
FC2D	FRE
FCCC	addition +
FCE1	soustraction -
FCF5	multiplication *
FD12	division /
FD37	division entière \
FD49	modulo (reste de la division).
FD58	fonction AND (ET LOGIQUE).
FD63	fonction OR (OU LOGIQUE).
FD6D	fonction XOR (OU EXCLUSIF).
FD85	ABS
FDE8	FIX
FDED	INT
FE8D	CINT
FEC2	UNT
FEBC	CREAL
FEF3	nettoyage de l'accumulateur
FF02	SGN
FF0A	positionnement d'un entier dans l'accumulateur
FF16	conversion en réel
FF1D	met le type de variable dans C

FF23	met le type de variable dans A
FF27	teste si l'accumulateur contient un pointeur de chaîne
FF62	copie l'accumulateur dans la zone pointée par DE
FF71	teste si majuscule
FF7B	teste si numérique
FF8A	conversion en majuscule
FFAA	compare A et le contenu de HL
FFB8	compare HL et DE
FFBE	compare HL et BC
FFC4	DE = HL - DE
FFCF	HL = HL - DE
FFDA	BC = HL - DE
FFE7	HL = HL - BC
FFF2	LDIR
FFF5	LDDR
FFF8	JP (HL)
FFF9	retour au contenu de BC
FFFB	retour au contenu de DE



LES INSTRUCTIONS MAL CONNUES DU BASIC AMSOFT.

7.1 Généralités.

Dans ce chapitre, nous allons passer en revue les instructions et les fonctions mal connues et donc mal aimées de la plupart des utilisateurs. Ce sont les fonctions en contact direct avec l'assembleur ou le matériel, celles qui référencient la mémoire interne et l'instruction fantastique mais ignorée DEF FN.

DEF FN - FN - MEMORY - HIMEM - FRE - PEEK - POKI
OUT - INP - WAIT - CALL - SYMBOL - SYMBOL AFTER
DI - EI - EVERY - AFTER - REMAIN

Nous nous attarderons plus particulièrement sur l'instruction CALL et les méthodes de chargement de programmes en langage machine par le BASIC.

** UNE DES FONCTIONS LES PLUS PUISSANTES DU BASIC AMSOFT **
** EST CERTAINEMENT LA FONCTION VARPTR SYMBOLISEE PAR @ **
** POURQUOI NE FIGURE T'ELLE PAS DANS LE MANUEL ??????? **
** - - - - NOUS ALLONS COMBLER CETTE LACUNE - - - - **

7.2 Instruction DEF FN et fonction FN.

SYNTAXE :

Déclaration : DEF FN NN(P1,P2,....PN) = Fonction BASIC

Utilisation : FN NN(p1,p2,....pn)

Comme des milliers d'utilisateurs du BASIC AMSOFT ou MICROSOFT, vous n'utilisez jamais les fonctions définies par l'utilisateur. Utiliser de telles fonctions n'apparaît pas immédiatement nécessaire au programmeur débutant et les exemples des manuels ne montrent pas leur utilité.

Pourtant, ces fonctions permettent des techniques de programmation particulièrement intéressantes.

Avantages :

- Les variables utilisées dans la fonction ne sont pas affectées par un appel.
- Les fonctions peuvent être définies n'importe où dans le programme, à condition que la logique du programme rencontre la définition au moins une fois avant un appel.
- On peut redéfinir une fonction autant de fois que nécessaire.
- On peut définir une fonction qui utilise d'autres fonctions définies.

INCONVENIENT :

- Une fonction ne peut pas contenir d'instruction.

EXEMPLES D'UTILISATION :

1° On veut réaliser une fonction qui donne la valeur hexadécimale d'une adresse mémorisée sous la forme classique (2 octets) à une autre adresse X.

- On peut écrire `AD$=HEX$(PEEK(X)+256*PEEK(X+1))`

- On peut aussi écrire :

```
DEF FN AD$(X)=HEX$(PEEK(X)+256*PEEK(X+1))
```

- Chaque fois qu'on devra faire appel à la fonction, il suffira d'écrire : `FN AD$(Z)` où Z est soit la valeur de l'adresse qui contient l'adresse à rechercher, soit une variable qui contient cette valeur.

2° On veut réaliser une fonction qui centre une chaîne de caractères dans un espace de N caractères.

- Il suffit d'écrire :

```
DEF FN CT$(A$,N)=STRING$(N/2-LEN(A$)/2-.5," ")+A$
```

- L'appel s'effectue grâce à la fonction : `FN CT$(Z$,I)` où Z\$ est soit la variable soit le texte à centrer, et I, le nombre de caractères du champs de centrage.

3° Calcul du jour courant dans l'année.

```
DEF FN JC(J,M,A)=(M-1)*28+VAL(MID$("000303060811131619212426",  
", (M-1)*2+1, 2))-((M>2)AND((A AND NOT -4)=0))+J
```

Remarque : Respectez les blancs entre `A AND NOT - 4`.

Utilisation : pour déterminer le numéro du jour courant dans l'année correspondant au 15 OCTOBRE 1985, écrire :

```
PRINT FN JC(15,10,1985) -> Réponse : 288
```

Le 15/10/85 est donc le 288 jour de l'année.

4° Calcul de la date interne.

```
DEF FN DT(J,M,A)=A*365+INT((A-1)/4)+(M-1)*28  
+VAL(MID$("000303060811131619212426", (M-1)*2+1, 2))  
-((M>2)AND((A AND NOT -4)=0))+J
```

Utilisation : Identique à l'exemple 3, mais le nombre fourni représente le nombre de jours écoulés depuis un référentiel. Cette fonction est valable pour toutes les dates comprises entre 1901 et 2099

5° Calcul du jour de la semaine.

Pour cette fonction, on utilise d'abord la précédente pour calculer la date interne, ensuite la date interne est transformée en un jour de semaine (LUNDI à DIMANCHE).

```
DEF FN JS$(N)=MID$("VENDREDISAMEDI..DIMANCHELUNDI...MARDI...  
MERCREDIJEUDI...", (N-INT(N/7)*7)*8+1, 8)
```

Remarque : En tapant cette phrase, remplacez les points par des blancs.

Utilisation : Quel jour tombe le premier janvier 1986 ?

A - calcul de la date interne :

```
PRINT FN DT(1,1,1986) -> 725387
```

B - calcul du jour :

```
PRINT FN JS$(725387) -> MERCREDI
```


7.3 Instruction MEMORY et fonctions HIMEM et FRE.

SYNTAXE :

Instruction : MEMORY valeur
Fonctions : X=HIMEM ou PRINT HIMEM
 : X=FRE(0) ou X=FRE(" ") ou PRINT FRE(0)...

A - MEMORY :

L'instruction MEMORY permet de définir la valeur supérieure de la mémoire au delà de laquelle le BASIC ne peut pas écrire pendant son exécution (mémorisation de variables) ou pendant l'encodage des lignes (mémorisation du programme). Bien entendu, cette instruction n'empêche pas le BASIC d'écrire dans ses zones de travail, même si celles-ci se trouvent à une adresse supérieure à la valeur donnée à MEMORY.

Le but de cette instruction est de protéger une zone mémoire afin de pouvoir y installer des programmes en langage machine et de s'assurer que ces programmes ne seront pas écrasés par le BASIC.

La valeur choisie doit en principe être proche de la l'adresse de départ des zones de travail. Si vous donnez une valeur trop faible à MEMORY, vos programmes BASIC ne pourront pas contenir plus de quelques lignes ou utiliser de nombreuses variables.

Si vous essayez de dépasser la valeur maximum donnée au départ (vous pouvez la consulter en tapant juste après l'initialisation : PRINT HIMEM), le message d'erreur MEMORY FULL s'affichera à l'écran et l'ordre ne sera pas pris en compte.

Exemple : MEMORY 44000 produit le message MEMORY FULL.

Si la valeur fournie est trop faible, le même message apparaît pour vous signaler qu'il n'y a pas assez de mémoire pour le BASIC.

Exemple : MEMORY 0 produit le message MEMORY FULL.

B - HIMEM

Cette fonction est très simple, elle fournit la valeur courante du paramètre de l'instruction MEMORY. Autrement dit, elle vous donne l'adresse mémoire supérieure utilisée par BASIC. Cette valeur ne se modifie pas en principe en cours de programme sauf si vous utilisez l'instruction SYMBOL AFTER. Cette fonction ne doit pas être confondue avec la fonction FRE(0) qui évolue sans cesse en cours de programme.

La fonction HIMEM permet de sauvegarder la valeur de départ de la mémoire et de rendre au BASIC tout son espace lorsqu'il n'a plus besoin de ses routines en langage machine.

```
10 X=HIMEM
20 MEMORY 10000
30 REM SUITE DU PROGRAMME
.....
9999 REM REMISE DE L'ESPACE TOTAL
10000 MEMORY X
```

Remarque : A propos de SYMBOL AFTER, une erreur existe dans la ROM de votre AMSTRAD. Essayez de modifier votre départ de mémoire par MEMORY (par exemple MEMORY 40000) et tapez ensuite SYMBOL AFTER suivi d'une valeur quelconque (par exemple SYMBOL AFTER 100). Le message IMPROPER ARGUMENT apparaît. La commande SYMBOL AFTER doit donc être utilisée avant l'instruction MEMORY. Est-ce un BUG ou une contrainte?

C - FRE

La fonction FRE se présente sous deux formes : Une forme FRE(0), et une forme FRE(""). L'argument 0 peut être remplacé par une valeur numérique quelconque ou par une variable numérique. L'argument "" peut être remplacé par une chaîne de caractères quelconque entre guillemets ou par une variable alphanumérique quelconque.

Cette fonction fournit une valeur identique dans les 2 cas. Cette valeur représente le nombre d'octets libres et utilisables par BASIC. Cette valeur évolue au cours de l'encodage du programme ainsi qu'au cours de son exécution. Elle diminue à chaque nouvelle variable rencontrée.

La première forme se contente de fournir la valeur de l'espace disponible. La deuxième force en plus l'ordinateur à exécuter un GARBAGE COLLECTION.

Le GARBAGE COLLECTION (littéralement ramassage de poubelle) consiste en une réorganisation de l'espace de travail des variables en supprimant les zones inutilisées et en "retassant" la mémoire. Cette fonction peut prendre un certain temps, mais elle permet en général de récupérer de l'espace.

Exemple :

Lancez le programme suivant :

```
10 DIM C$(50),D$(50)
20 FOR I=1 TO 50
30 FOR J=1 TO 200
40 C$(I)=C$(I)+"A"
50 D$(I)=D$(I)+"B"
60 NEXT J
70 NEXT I
```

Patientez quelques instants.

Tapez PRINT FRE(0) : vous obtenez l'espace restant.

Tapez PRINT FRE("") : vous obtenez un nombre plus grand qui est le nouvel espace restant. L'espace regagné est d'environ 1000 octets et cette deuxième forme a pris une petite seconde avant de vous donner un résultat.

7.4 Instruction POKE et fonction PEEK.

SYNTAXE :

Instruction : POKE adresse, valeur
fonction : X=PEEK(adresse) ou PRINT PEEK(adresse)

A - POKE

Cette instruction permet d'écrire dans la mémoire de façon violente.

L'adresse peut être comprise entre 0 et 65535 (0 à FFFF hexa). La valeur peut être comprise entre 0 et 255 (8 bits).

Exemple : POKE 40000,87 : écrit la valeur 87 à l'adresse 40000.

La mémoire étant complètement constituée de RAM, vous pouvez écrire dans n'importe quelle portion de celle-ci.

Cependant, l'utilisation de cette instruction est particulièrement dangereuse. Une utilisation inconsidérée peut "planter" le système ou encore détruire le contenu d'une disquette (CPC664).

Malgré cette restriction, l'instruction POKE peut faire des miracles et vous trouverez peu de programmes BASIC dans ce livre qui ne l'utilise pas.

En guise de hors d'oeuvre, nous allons modifier dynamiquement un programme BASIC au moyen de l'instruction POKE.

Encodez le programme suivant : 10 A\$="COUCOU"

Tapez POKE 375,193

Listez le programme, il est devenu : 10 B\$="COUCOU"

Nous avons modifié la mémoire 375 qui contenait 192 (A) et nous avons remplacé son contenu par 193 (B).

Pour pouvoir écrire dans la mémoire sans problème, il faut choisir une zone inutilisée par le BASIC et la protéger au moyen de l'instruction MEMORY.

Il est parfois utile de modifier la région de communication ou la table des variables système par des POKES. Nous verrons quelques exemples de cette utilisation au chapitre réservé aux trucs et astuces.

B - PEEK

La fonction PEEK permet de lire le contenu d'une mémoire.

Cette fonction ne subit pas les restrictions de l'instruction POKE. Vous pouvez donc aller lire le contenu de toutes les mémoires sans danger.

Cette fonction est très utile pour pouvoir contrôler le contenu des différents paramètres de la table des variables système. Les exemples d'application sont multiples. En voici un choisi parmi tant d'autres :

Pour contrôler le mode trigonométrique courant, faire :

```
PRINT PEEK(&B8F7) (CPC464)
PRINT PEEK(&B113) (CPC664)
```

Une valeur égale à 0 indique le mode RADIAN, une valeur égale à 255 indique le mode DEGRE.

7.5 Instructions OUT et WAIT et fonction INP.

A - OUT

SYNTAXE : OUT adresse,valeur

L'instruction OUT permet d'écrire sur un port d'entrée/sortie périphérique du microprocesseur Z80. l'adresse du PORT est, contrairement aux autres systèmes (MSX,SINCLAIR,TANDY...), exprimée sur 16 bits, ce qui permet de disposer de 65536 ports. A moins de disposer de matériels particuliers, seuls quelques ports sont utilisables (voir chapitre 1). La valeur à écrire sur le port est un entier de 8 bits (0 à 255).

Cette instruction, comme l'instruction POKE, doit être utilisée avec circonspection. La moindre erreur "plante" le système ou détruit la disquette.

B - INP

SYNTAXE : X=INP(adresse) ou PRINT INP(adresse)

Cette fonction fournit la valeur courante du port dont on spécifie l'adresse. Comme dans l'instruction OUT, l'adresse est exprimée sur 16 bits et permet d'adresser 65536 ports. Cette fonction est sans danger pour le système. Cependant, peu de ports ont une fonction en lecture, l'utilisation de la fonction INP est donc des plus réduites.

SYNTAXE : WAIT adresse, octet de masque, octet de sélection

L'instruction WAIT, comme la fonction INP, lit le contenu du port dont l'adresse est spécifiée. Ensuite, elle applique à la valeur lue une fonction ET logique avec l'octet de masque pour isoler les bits à tester. Enfin, elle applique une fonction OU EXCLUSIF avec l'octet de sélection pour inverser l'état de certains bits. Cette instruction ne rend la main au programme que lorsque le résultat de ces différentes opérations est différent de 0.

Exemple : Je désire attendre tant que le bit le plus significatif (B7) du port F500 (PPI port B) est 0.

Il suffit d'écrire WAIT &F500, &X10000000 ou encore
WAIT &F500, 128

Si je désire l'inverse, c'est à dire attendre tant que le bit 7 du port est différent de 0, je dois inverser le bit 7 et écrire : WAIT &F500, &X10000000, &X10000000 ou encore :
WAIT &F500, 128, 128

7.6 Instruction CALL.

7.6.1 INTRODUCTION

SYNTAXE : CALL adresse[,liste de paramètres]

ou liste de paramètres est une zone optionnelle qui peut être composée de 32 paramètres maximum séparés par une virgule.

Un paramètre peut être :

- une constante entière (-32768 à 32767 ou 0 à FFFF)
- une variable contenant une valeur entière ou une chaîne de caractères.
- un pointeur de variable @ (voir VARPTR).

Cette instruction permet l'appel à un sous-programme externe au BASIC et écrit en langage machine. Ce sous-programme doit avoir son point d'entrée (le premier octet exécutable) à l'adresse spécifiée dans l'instruction CALL.

Le BASIC est un langage facile à utiliser et très efficace pour les calculs mathématiques et les programmes de gestion. Mais lorsqu'une exécution ultra-rapide ou une économie de mémoire est nécessaire, on doit s'adresser au processeur dans sa langue maternelle : le langage machine.

A l'exception des jeux d'arcades, il est rarement pratique d'écrire un programme complet en langage machine, cette écriture étant fastidieuse et longue.

La meilleure approche consiste à réaliser le programme en BASIC et à programmer les sous-routines trop longues au point de vue temps en langage machine.

L'utilisation de sous-programmes en langage machine dans un programme BASIC nécessite quelques précautions.

- A - Interdire au BASIC et à ses tables de rentrer en conflit avec lui au point de vue de l'emplacement. Ceci est résolu par l'instruction MEMORY.

- B - Introduire (charger) le programme en langage machine dans la mémoire. Nous analyserons dans la suite de cette section les différentes façons de procéder.

- C - Définir l'adresse de départ de la routine. En général, c'est le premier octet de la routine, mais cette règle n'est pas absolue.

- D - Exécuter la routine. C'est le but de la fonction CALL.

7.6.2 FONCTIONNEMENT

Après l'appel de l'instruction CALL, le registre PC du processeur est positionné à l'adresse indiquée dans l'instruction. Autrement dit, la mémoire "adresse" est accédée pour y trouver une instruction exécutable.

A ce moment, le registre A du Z80 contient le nombre de paramètres passés à la routine. Ce nombre ne peut pas dépasser 32. le registre DE contient le dernier paramètre et le registre IX pointe sur une zone mémoire où sont disposés tous les paramètres. Le registre IX pointe sur le dernier paramètre.

Chaque paramètre est constitué de 2 octets. Ces octets sont présentés sous la forme habituelle en Z80, poids faible d'abord, poids fort ensuite.

Si le paramètre est un entier ou une variable contenant un entier, c'est la valeur de cet entier en binaire signé qui se trouve dans la mémoire pointée par IX.

Si le paramètre est une variable alphanumérique, c'est l'adresse du VARPTR que l'on retrouve dans la mémoire pointée par IX.

Si le paramètre est un pointeur de variable (VARPTR), c'est le pointeur de la variable (2 octets) que l'on retrouve dans la mémoire pointée par HL.

Il est donc identique de passer une variable alphanumérique par elle même ou par son pointeur.

```
10 A$="COUCOU" : CALL &7000,A$ est identique à  
10 A$="COUCOU" : CALL &7000,@A$
```

Pour fournir une valeur à une variable BASIC en sortant

de la routine, il suffit d'initialiser la variable avant d'entrer dans la routine et de passer son pointeur à la routine. Cette technique sera étudiée en détails par la suite.

Nous avons dit que IX pointe sur le dernier paramètre. Chaque paramètre étant composé de 2 octets, IX + 2 pointe sur l'avant dernier paramètre, IX+4 sur le précédent et ainsi de suite.

Exemple :

```
A=1  
B$="TOTO"  
CALL adresse,&l234,A,@A,B$,@B$
```

Il y a 5 paramètres : le registre A du processeur contient 5

IX + 0 : pointe sur l'octet de poids faible de l'adresse du pointeur de B\$.
IX + 1 : pointe sur l'octet de poids fort de l'adresse du même pointeur.
IX + 2 : pointe comme IX + 0.
IX + 3 : pointe comme IX + 1.
IX + 4 : pointe sur l'octet de poids faible de l'adresse de la valeur de A.
IX + 5 : pointe sur l'octet de poids fort de l'adresse de la valeur de A.
IX + 6 : pointe sur 1 : poids faible de 1.
IX + 7 : pointe sur 0 : poids fort de 1.
IX + 8 : pointe sur 34h : poids faible de &l234.
IX + 9 : pointe sur 12h : poids fort de &l234.

7.7 Chargement d'un programme en langage machine

7.7.1 METHODE DATA ET POKE

C'est la méthode la plus simple. Elle est utilisée par tous les exemples de ce manuel. Ce n'est pas la plus rapide ni la plus économique.

Procédure : Il suffit de mettre les valeurs à introduire en mémoire (en décimal ou en hexadécimal) dans les lignes de DATA, ensuite on les lit et on les envoie en mémoire par des POKE successifs et ce, au moyen d'une boucle FOR - NEXT .

Exemple : Ecrire les 5 octets 62,255,62,0 221 à l'adresse 10000.

```
10 MEMORY 9999
20 FOR I=10000 TO 10004
30 READ A
40 POKE I,A
50 NEXT I
60 DATA 62,255,62,0,221
```

Le même exemple mais avec des valeurs hexadécimales donne :

```
10 MEMORY 9999
20 FOR I=10000 TO 10004
30 READ A$
40 POKE I,VAL("&" + A$)
50 NEXT I
60 DATA 3E,FF,3E,00,C9
```


7.7.2 METHODE DE LA CHAINE DE CARACTERES

Cette méthode, très en vogue en BASIC Microsoft, s'applique assez difficilement à l'AMSTRAD, mais elle n'est pas impossible à mettre en oeuvre.

Avantage de la méthode :

- Le MEMORY n'est pas nécessaire.
- Le déplacement est facile.
- Le temps de chargement est nul.

Inconvénients :

- Routine limitée à 255 caractères.
- Pas d'octet à 0 dans la routine.
- Le programme doit être indépendant de l'adresse d'implantation.

Description de la méthode :

Encoder une première ligne de programme constituée d'une variable alphanumérique composée d'autant d'* qu'il y a d'octets à installer.

Trouver l'adresse du premier * (avec des PEEK par exemple).

Installer la routine par des POKE successifs.

La routine étant appelée par un CALL à l'adresse :

ADRESSE = PEEK(@Z\$+1)+256*PEEK(@Z+2)

où Z\$ représente le nom de la variable qui contient les * au départ.

Exemple :

```
10 Z$="*****"           : REM 5 OCTETS
```

Si 378 est l'adresse de la première étoile, faites :

```
50000 FOR I=378 TO 382
50010 READ X
50020 POKE I,X
50030 NEXT I
50040 DATA 62,255,62,64,221
```

Lancer le programme à la ligne 50000

Effacer les lignes 50000 à 50040

```
Faire 20 AD=PEEK(@Z$+1)+256*PEEK(@Z$+2)
      30 CALL AD
```


7.7.3 METHODE DE LA VARIABLE TABLEAU

On peut charger une routine en langage machine dans une variable tableau entière.

AVANTAGES :

- Pas de MEMORY nécessaire.
- Le transfert d'arguments est très facile.
- On peut utiliser des octets égaux à 0.

INCONVENIENT :

- Le programme doit être indépendant de l'adresse.

DESCRIPTION DE LA METHODE :

- Définir la variable tableau comme variable entière.
- Diviser le nombre d'octets de la routine par 2 et prendre le plus grand entier - 1.
- Dimensionner la variable avec la valeur ainsi trouvée
- Calculer la valeur de chaque élément du tableau en utilisant la formule suivante :

$$X = \text{octet}(n) + 256 * \text{octet}(n+1)$$

- Etablir les égalités d'éléments.
- Définir le point d'entrée au moment du CALL par :

CALL @N(0)

ou N est le nom de la variable tableau.

Exemple : Soit le programme de 5 octets suivants :

3E,10,3E,40,C9

```
10 DEFINT A : REM Variable A entière
20 DIM A(2) : REM 2 = INT(5/2+1)-1
30 A(0)=4158 : REM 4158 = 16 (10H) * 256 + 62 (3EH)
40 A(1)=16446: REM 16446= 64 (40H) * 256 + 62 (3EH)
50 A(2)=221 : REM 221 = 0 * 256 + 221(C9H)
60 CALL @A(0)
```

Remarque : Si une valeur est supérieure à 32767, il faut lui soustraire 65536.

Exemple : 3ED2

$A(n) = 210 \text{ (D2H)} * 256 + 62 \text{ (3EH)} = 53822$

$53822 > 32767 \rightarrow A(n) = 53822 - 65536 = -11714$

D'autres possibilités existent. On peut charger un programme en langage machine dans une ligne de remarque, ou avant le début du BASIC en modifiant le pointeur de début de programme

7.8 Instructions SYMBOL et SYMBOL AFTER.

SYNTAXE : SYMBOL n,R1,R2,R3,R4,R5,R6,R7,R8
 SYMBOL AFTER m

Où n,m et R1 à R8 sont des entiers compris entre 0 et 255.

A - SYMBOL AFTER

L'instruction SYMBOL AFTER a pour effet de copier une partie du générateur de caractères contenu dans la ROM vers la RAM. Le paramètre m indique le numéro du caractère à partir duquel la copie est effectuée. En standard à l'initialisation, les 16 derniers caractères sont copiés dans la RAM. Une instruction équivalente à SYMBOL AFTER 240 est donc exécutée à l'initialisation (il y a 256 caractères au total).

Le paramètre m peut prendre toutes les valeurs entre 0 et 255.

Evidemment, la place prise par le générateur de caractères en RAM l'est au détriment du BASIC. Cette instruction "consomme" 8 octets par caractère défini. Ainsi, entre SYMBOL AFTER 255 (pas de caractère défini) et SYMBOL AFTER 0 (tous les caractères sont définis en RAM), la mémoire disponible en BASIC est "amputée" de 2048 octets. Ce que vous pouvez vérifier aisément au moyen de PRINT FRE(0).

La zone mémoire réservée par cette instruction est située en haut de mémoire.

L'instruction SYMBOL permet de redéfinir un caractère. Le code ASCII du caractère à redéfinir est donné par le paramètre n. Chaque matrice de caractère est définie sur 8 x 8 points. La nouvelle matrice du caractère est définie par les 8 paramètres R1 à R8. Chaque paramètre Ri correspond à une ligne horizontale de la matrice de caractères. Chaque paramètre définit une séquence de 8 points horizontaux (8 bits). Les bits à 1 identifient les points à afficher, les bits à 0 identifient les points "transparents".

Exemple : définir le caractère suivant à la place du A.

.....	:	11111111	:	FF	:	255
.	:	10011001	:	99	:	153
.	:	10011001	:	99	:	153
.	:	01100110	:	33	:	51
.	:	10000001	:	81	:	129
.....	:	01111110	:	7E	:	126
.	:	11000011	:	C3	:	195
... -..	:	11100111	:	E7	:	231

A est le caractère de code ASCII 41H ou 65.

Il suffit d'écrire

SYMBOL AFTER 64

SYMBOL 65,255,153,153,51,129,126,195,231

7.9 Les instructions de gestion des interruptions

7.9.1 ETUDE THEORIQUE DES INTERRUPTIONS

D'un point de vue strictement matériel les interruptions sont synchronisées par le signal de retour du SPOT en provenance du CRTIC. Ce signal est appliqué sur la broche d'interruption du Z80. Ces interruptions se produisent tous les 1/300 ième de seconde.

Les interruptions peuvent être divisées en 4 types:

A - L'interruption primaire : 1/300 seconde. C'est la plus rapide, elle n'est pas prévue pour un usage courant.

B - L'interruption pour le PSG : 1/100 seconde. C'est l'interruption qui est utilisée pour la maintenance des files d'attentes (queues) sonores.

C - L'interruption de retour du spot : 1/50 seconde (fréquence réseau). Elle est utilisée pour effectuer des actions précises pendant la période de non affichage de l'image (changement de couleur ...).

D - L'interruption à usage général : 1/50 seconde. C'est elle qui nous intéresse le plus. Elle est utilisée pour scruter le clavier et pour incrémenter les 4 horloges contenues dans le système.

Pour qu'une même interruption puisse agir sur plusieurs compteurs séparés possédant chacun des caractéristiques propres, le système de base a la faculté de traiter des files d'attentes d'interruptions. Ces files sont associées à des classes d'événements. Une théorie complète de cette méthode serait trop aride pour le présent ouvrage. Nous conseillons au lecteur intéressé et anglophile, la lecture du COMPLETE FIRWARE SPECIFICATION SOFT 158.

A - DI et EI

SYNTAXE : DI (DISABLE INTERRUPT)
EI (ENABLE INTERRUPT)

Ces deux instructions permettent respectivement d'interdire ou d'autoriser les interruptions en provenance du gestionnaire d'interruptions (compteurs).

On utilise ces deux instructions pour interdire l'interruption d'une sous-routine par des événements en provenance des compteurs (voir EVERY & AFTER).

Remarque : La commande BREAK (qui fonctionne par interruption) n'est pas affectée par ces instructions.

B - AFTER et EVERY

SYNTAXE : AFTER n[,m] GOSUB ligne
EVERY n[,m] GOSUB ligne

Ou n est un nombre entier, m est un nombre compris entre 0 et 3 et ligne le numéro de ligne de la sous-routine.

Ces deux instructions lancent l'exécution d'une sous-routine BASIC après un laps de temps déterminé par la valeur de n (n exprime le nombre de 1/50 ième de seconde avant le lancement). Le numéro du compteur à prendre en compte est représenté par le paramètre optionnel m. Si ce paramètre est omis, le chronomètre (compteur) 0 est pris en compte.

La différence entre AFTER et EVERY est la suivante : AFTER ne lance la routine qu'une seule fois à l'issue du laps de temps spécifié alors qu'EVERY réarme le compteur à chaque lancement.

SYNTAXE : X=REMAIN(m) ou PRINT REMAIN(m)

REMAIN est une fonction qui permet de connaître le laps de temps qui reste à accomplir sur le chronomètre m avant le lancement d'un sous-programme. Cependant, cette commande arrête le chrono spécifié (pourquoi ?). L'appel de la sous-routine ne peut donc plus avoir lieu après la consultation du temps qui reste à accomplir.

7.10 La fonction @ (VARPTR).

SYNTAXE : X=@var ou PRINT @var

Ou var représente une variable quelconque.

7.10.1 GENERALITES

La fonction @ (VARPTR) est un des plus merveilleux outils du BASIC. Elle permet d'atteindre l'adresse de stockage des valeurs assignées aux variables ainsi que différentes informations sur leur contenu.

A l'aide des adresses obtenues par la fonction @, de l'instruction POKE et de la fonction PEEK, on peut effectuer une foule d'opérations très utiles.

L'utilisation principale de la fonction @ est certainement de retrouver des informations sur les chaînes de caractères.

Lorsqu'on écrit : 10 A\$="COUCOU", le système d'exploitation de l'interpréteur BASIC doit sauvegarder la valeur affectée à A\$ (en l'occurrence COUCOU) quelque part dans la mémoire (voir section sur l'espace réservé aux variables).

Lorsque, quelques lignes plus bas, on écrit : 50 PRINT A\$, le système devra être capable de retrouver COUCOU.

Pour effectuer cette opération, le BASIC possède une liste des variables utilisées. Chaque fois qu'il rencontre une nouvelle variable, il l'ajoute à la liste.

La variable qui a été rencontrée la première dans le programme sera la première dans la liste, et celle qui sera rencontrée la dernière dans le programme, sera la dernière dans la liste.

Chaque fois que le BASIC rencontre une nouvelle variable, il fouille la liste pour voir si cette variable a déjà été affectée. Si ce n'est pas le cas, il l'ajoute à la liste.

Le BASIC possède deux listes : une pour les variables simples, et une pour les variables dimensionnées. Le système consulte la liste correspondant à la variable rencontrée.

Remarque Le temps pris par le système pour retrouver une variable est un facteur influençant très fort la vitesse d'exécution des programmes. Il est possible d'améliorer de façon notable la vitesse d'exécution d'un programme en définissant au début du programme une liste des variables les plus souvent utilisées.

Les variables simples sont définies la première fois qu'on leur attribue une valeur, les variables tableau sont définies lors de l'instruction DIM.

En plus du nom de la variable, la liste contient des informations sur le type de variable.

En fonction du type, d'autres informations sont fournies au système : soit la valeur de la variable, soit l'adresse où l'on peut trouver cette variable.

Le BASIC utilise ces informations pour retrouver rapidement les valeurs des variables lors de l'exécution d'un programme.

Toutes ces informations sont aisées à accéder grâce à la fonction @ (VARPTR pour VARIABLE Pointeur).

7.10.2 UTILISATION DE @

L'instruction `X=@A$` fournira une valeur X, adresse où des informations sur A\$ pourront être trouvées.

La variable sur laquelle on demande des renseignements peut être une variable entière, une variable flottante, une variable alphanumérique ou un élément d'une variable tableau de n'importe quel type

Exemple : `X=@A$(2)` est parfaitement licite.

L'utilisation que l'on peut faire de l'adresse contenue dans X à l'issue de l'instruction est fonction du type de la variable.

La valeur contenue dans X étant une adresse, elle est comprise entre 0 et 65535. Donc c'est une valeur entière et

elle tient sur 2 octets.

Contenu de AD dans la fonction AD=@var.

Le contenu de l'adresse fournie par la fonction @ varie en fonction du type de variable

Si AD est l'adresse fournie par la fonction @ alors quel que soit le type de la variable :

l'adresse AD-1 contient le type de la variable :

1 = Variable entière.

2 = Variable alphanumérique.

4 = Variable flottante.

L'adresse AD-2 contient le dernier caractère du nom de la variable avec le bit 7 à 1 (augmenté de 128).

De AD-3 à AD-n on trouve les autres caractères du nom de la variable s'ils existent.

Exemple : Si ANNE est une variable chaîne qui contient "TOTO" alors :

AD-5	AD-4	AD-3	AD-2	AD-1	
A	N	N	.	.	(ASCII)
41	4E	4E	C5	2	(HEXA)

Le contenu des adresses AD et suivantes, dépend du type de la variable. Mais quelque soit le type de variable, le nombre d'octets à considérer vaut l'indicateur de type (AD-1) augmenté de 1. Ainsi si la variable est entière, il faut considérer 2 octets (1+1).

Si la variable est :
=====

1° Une variable entière : -----

Une variable entière est mémorisée sur 16 bits en binaire signé. Le bit le plus significatif (B15) est le bit de signe.

AD contient les 8 bits les moins significatifs de la variable (B0 à B7).

AD+1 contient les 8 bits les plus significatifs de la variable (B8 à B15).

Exemples :

a) Si la variable entière vaut 12345 décimal (3039 en hexa), alors :

L'adresse AD contient 57 (39H) et AD+1 contient 48 (30H)

Et $57 + 256 * 48 = 12345$

Faites :

```
10 DEFINT A
20 A=12345
30 AD=@A
40 PRINT PEEK(AD);PRINT(AD+1)
50 PRINT PEEK(AD)+256*PEEK(AD+1)
```

b) Si la variable vaut -12345 (CFC7H)

L'adresse AD contient 199 (C7H) et AD+1 contient 207 (CFH)

Et $199 + 256 * 207 - 65536 = -12345$

2° UNE VARIABLE FLOTTANTE

Les 5 octets de AD à AD+4 contiennent la valeur de la variable.

La façon de représenter une variable en 5 octets est particulièrement complexe à expliquer. Essayons par quelques exemples de comprendre le mécanisme. Les valeurs données pour AD0 à AD4 sont en hexadécimal, la valeur de la variable est en décimal.

var	AD+0	AD+1	AD+2	AD+3	AD+4	
0	0	0	0	0	0	FACILE
1	0	0	0	0	81	DEJA PLUS COMPLEXE
2	0	0	0	0	82	COMPRIS ?
8	0	0	0	0	84	OU ?
-8	0	0	0	30	84	BIZARRE
-16	0	0	0	B0	85
0.5	0	0	0	0	80	
3	0	0	0	40	82	
2.5	0	0	0	20	82	

Constatation :

- 0 est une exception facile à comprendre.
- Le bit B7 de AD+3 indique le signe 0=positif, 1=négatif
- AD+4 - 81H représente la puissance de 2 pour obtenir le nombre inférieur le plus proche du nombre cherché. ($8 = 2^3$ (84-81)). Autrement dit si le nombre contenu dans var est m alors AD+4 contient le logarithme en base 2 de m augmenté de 81H.
- Les autres bits doivent être lus en partant de B6 de AD+3 jusqu'à B0 de AD. Ils indiquent la fraction du nombre représenté par AD+4 qui doit être ajoutée pour obtenir le nombre désiré. B6 de AD+3 vaut 1/2, B5 vaut 1/4 ... B0 vaut 1/64 et ainsi de suite avec les bits de AD+2, AD+1 et AD.

Exemples :

A) Si AD+4 contient 88H et AD+3 contient 60H alors le nombre est :

2 exposant 88-81 soit 2 exposant 7 = 128

$AD+3 = 60 = 01100000$ c-à-d $+ 1/2 + 1/4$ de 128 autrement dit
 $128 + 64 + 32 = 224$

B) Représentons le nombre -12345

- négatif : B7 de AD+3 = 1
- LOG base 2 de 12345 = 13 ou 0D -> PRINT LOG(12345)/LOG(2)
- 2 exposant 13 = 8192
- AD+4 contient donc 81H + 0DH = 8EH
- Exprimez le nombre en binaire -> PRINT BIN\$(12345)
11000000111001
- Mettre à 0 le premier digit significatif.
01000000111001
- Complétez par des 0 à droite pour obtenir un multiple de 8 bits. Coupez par tranche de 8 bits.
01000000 11100100
- Vous obtenez les valeurs respectives de AD+3 , AD+2 ...
- Mettre le bit de signe de AD+3 (B7)
11000000 11100100
-> AD+3 = C0 , AD+2 = E4 , AD+1 = 0 et AD = 0

3° UNE VARIABLE ALPHANUMERIQUE

AD contient la longueur de la chaîne en nombre d'octets.

AD+1 contient la valeur basse de l'adresse à laquelle on trouve le contenu de la variable.

AD+2 contient la valeur haute de cette même adresse.

4° UNE VARIABLE TABLEAU

La partie donnée d'une variable tableau est mémorisée de façon identique à celle d'une variable simple. Les éléments sont simplement disposés les uns après les autres en commençant par l'élément d'indice 0.

Si la variable a une seule dimension (vecteur) :

Si AD est l'adresse du pointeur de l'élément var(0).

Alors :

AD-1 et AD-2 contiennent le nombre d'éléments du vecteur. Autrement dit, la dimension + 1. Ce nombre codé sur 16 bits est présenté avec l'octet le moins significatif en AD-2 et l'octet le plus significatif en AD-1.

AD-3 contient la dimension de la variable (1).

AD-4 et AD-5 contiennent le nombre d'octets à ajouter à AD-4 pour arriver à la variable suivante (OFFSET). Autrement dit AD-4 et AD-5 contiennent $3 + N * L$ avec L égal au nombre d'éléments et N égal à 2, 3 ou 5 suivant le type de variable. AD-4 contient l'octet le plus significatif de ce nombre et AD-5 contient l'octet le moins significatif.

AD-6 contient le type de variable (1, 2 ou 4).

AD-7 contient la valeur ASCII de la dernière lettre du nom de la variable augmentée de 80H.

AD-8 contient l'avant dernière lettre du nom de la variable et ainsi de suite.

Exemple variable entière ALBA dimensionnée à 8 : DIM ALBA(8)

AD-10	AD-9	AD-8	AD-7	AD-6	AD-5	AD-4	AD-3	AD-2	AD-1	AD	AD+1
A	L	B
41	4C	42	C1	1	13	0	1	9	0	valeur 0	

Si la variable est multidimensionnée :

La structure est la suivante :

AD-n contient la valeur ASCII du dernier caractère du nom de la variable augmentée de 80H.

AD-n+1 contient le type de la variable

AD-n+2 et AD-n+3 contiennent la longueur de l'OFFSET ($1 + 2 * \text{nombre d'indices} + N * L$).

AD-n+4 contient le nombre d'indices.

AD-n+5 et AD-n+6 contiennent le nombre d'éléments du premier indice

AD-n+7 et AD-n+8 contiennent le nombre d'éléments du deuxième indice et ainsi de suite.

REMARQUES : Pour déterminer n il suffit de faire $6 + 2 * \text{nombre d'indices}$. La description des variables tableaux à une dimension est un simple cas particulier de la description des variables multidimensions.

LES R.S.X.

8.1 Généralités.

Voici un nom barbare pour un chapitre qui fait des miracles. R.S.X. est un acronyme anglo-saxon pour RESIDENT SYSTEM EXTENSION. Point n'est besoin de revenir d'Oxford pour traduire ceci en EXTENSION RESIDENTE DANS LE SYSTEME.

Les R.S.X. servent à ajouter de nouvelles commandes au langage BASIC. Par exemple : une commande de traçage de cercle qui fait cruellement défaut sur l'AMSTRAD. Ces R.S.X. sont écrits complètement en assembleur suivant un canevas bien déterminé qui sera expliqué en détail à la section 8.2.

Toutes les commandes BASIC (PRINT, GOTO, FOR..) sont appelées des commandes INTERNES. Quand un programme BASIC en cours d'exécution rencontre une de ces commandes, l'interpréteur "fouille" une table contenue dans la ROM à la recherche du mot spécifié (cette table se trouve à l'adresse E388H dans le CPC 464). Si la commande existe, le BASIC transfère l'exécution à une adresse bien définie qui dépend du mot rencontré. Si une commande externe est rencontrée, la RAM est "fouillée" de la même façon que la ROM pour déterminer si cette commande existe.

Une commande externe est constituée d'une chaîne de caractères alphanumériques précédée du signe : (obtenu en poussant en même temps sur SHIFT et sur 0).

Exemples : :CERCLE , :CARRE , :REMPLIR

Si vous tapez :CARRE dans votre programme à l'heure actuelle vous obtiendrez un éloquent "UNKNOWN COMMAND" qui signifie COMMANDE INCONNUE. L'interpréteur BASIC ne peut pas trouver la commande spécifiée dans sa ROM ou sa RAM. Il faut donc la définir et signaler à l'interpréteur qu'elle existe. Voici comment procéder.

8.2 Constitution d'un R.S.X.

La première étape consiste à avertir l'ordinateur de l'existence d'une table de commande(s) extérieure(s). Cette table fonctionne comme un pointeur indirect vers les routines de traitements propres à la commande. Ces traitements spécifiques se chargeront du contrôle de syntaxe et de l'exécution de la commande.

Cette opération se fait à l'aide d'une routine interne de la ROM. Cette routine est appelée par le vecteur situé à l'adresse BCD1.

=====

ROUTINE : KL LOG EXT	adresse BCD1
-----	-----

Conditions d'entrée : BC pointe sur la table des sauts d'extension de commande, HL pointe sur une zone libre de 4 octets en RAM.

Conditions de sortie : DE est modifié.

=====

Analysons le programme théorique.

	ORG	xxxx	Adresse d'implantation routine
TAMPON:	DEFS	4	Réserve 4 octets pour HL
DEBUT:	LD	BC, COMEXT	BC pointe sur table
	LD	HL, TAMPON	HL pointe sur zone de 4 octets
	CALL	#BCD1	Appel de la routine système
	RET		Retour au BASIC
COMEXT:	----		Début table des sauts

Ensuite, il faut définir l'adresse de début de la table des noms de commandes suivi d'autant d'instructions de saut absolu qu'il y a de commandes.

Exemples :

1° Cas d'une commande seule.

COMEXT:	DEFW	TABLE	Définition table des noms
	JP	CARRE	Saut au traitement spécifique
TABLE:	----		Début de la table des noms

2° Cas de plusieurs commandes.

COMEXT:	DEFW	TABLE	
	JP	CARRE	Saut traitement CARRE
	JP	CERCLE	Saut traitement CERCLE
	JP	REEMPLIR	Saut traitement REEMPLIR
	
	
TABLE:	----		Début de la table des noms

Ensuite, il faut définir la table des noms des nouvelles commandes. Elle est composée des caractères ASCII du mot clé représentant la commande sans le premier caractère : . Le dernier caractère du mot doit avoir son bit 7 à 1, autrement dit, on doit ajouter 80H à la valeur ASCII du dernier caractère du mot. Le dernier caractère de la dernière commande doit être suivi d'un octet à 0.

Exemple :

1° Cas d'une seule commande.

TABLE:	DEFM	"RECTANGL"	Mot RECTANGL
	DEFB	"E"+#80	Lettre E + 80H (bit 7=1)
	DEFB	0	Fin de table

TABLE	DEFM	"CARR"	MOT CARRE
	DEFB	"E"+#80	
	DEFM	"CERCL"	MOT CERCLE
	DEFB	"E"+#80	
	DEFM	"REPLI"	MOT REMPLIR
	DEFB	"R"+#80	
	DEFB	0	FIN DE TABLE

Enfin, à la suite de tout cela, il vous reste à écrire la routine spécifique de traitement.

Le programme complet (à l'exception de la routine de traitement) pour une commande de traçage de rectangle s'écrira :

A000					ORG	#A000
A000	00	00	00	00	TAMPON: DEFS	4
A004	01	0E	A0		DEBUT: LD	BC, COMEXT
A007	21	00	A0		LD	HL, TAMPON
A00A	CD	D1	BC		CALL	#BCD1
A00D	C9				RET	
A00E	13	A0			COMEXT: DEFW	TABLE
A010	C3	1D	A0		JP	RECTG
A013	52	45	43	54	TABLE: DEFM	"RECTANGL"
	41	4E	47	4C		
A01B	C5				DEFB	"E"+#80
A01C	00				DEFB	00
A01D	--	--	--	--	RECTG: ----	----- SUITE

8.3 Programmation du traitement d'une commande.

Lors de l'appel d'un R.S.X., le système se branche à l'adresse de traitement du mot clé rencontré (dans l'exemple ci-dessus, lors de la rencontre du mot RECTANGLE, le système se branche à l'adresse A01D). A cet instant, comme dans l'instruction CALL décrite au chapitre 7, le registre A du processeur Z80 contient le nombre d'arguments de la commande (paramètres) et le registre IX pointe sur le dernier paramètre.

La structure des paramètres dépend du type de paramètre.

- Si le paramètre est un entier, il est composé de 2 octets représentant l'entier en binaire signé (complément à 2).
- Si le paramètre est un réel, il est composé de 2 octets représentant le réel transformé en entier non signé.
- Si le paramètre est une variable, il est composé de 2 octets contenant la valeur de la variable. Remarque : si la variable est du type caractère c'est l'adresse du descripteur de la variable qui est fourni.

Il faut noter que la valeur est mémorisée avec l'octet le moins significatif en premier lieu comme c'est souvent le cas en Z80.

En résumé, chaque paramètre occupe 2 octets.

Exemple :

Supposons notre commande constituée de 3 paramètres valant respectivement 10, 300 et -1.

A l'entrée dans la routine de traitement, l'accumulateur contient 3 (le nombre de paramètres) et IX pointe sur une adresse mémoire qui contient la valeur -1 en binaire signé.

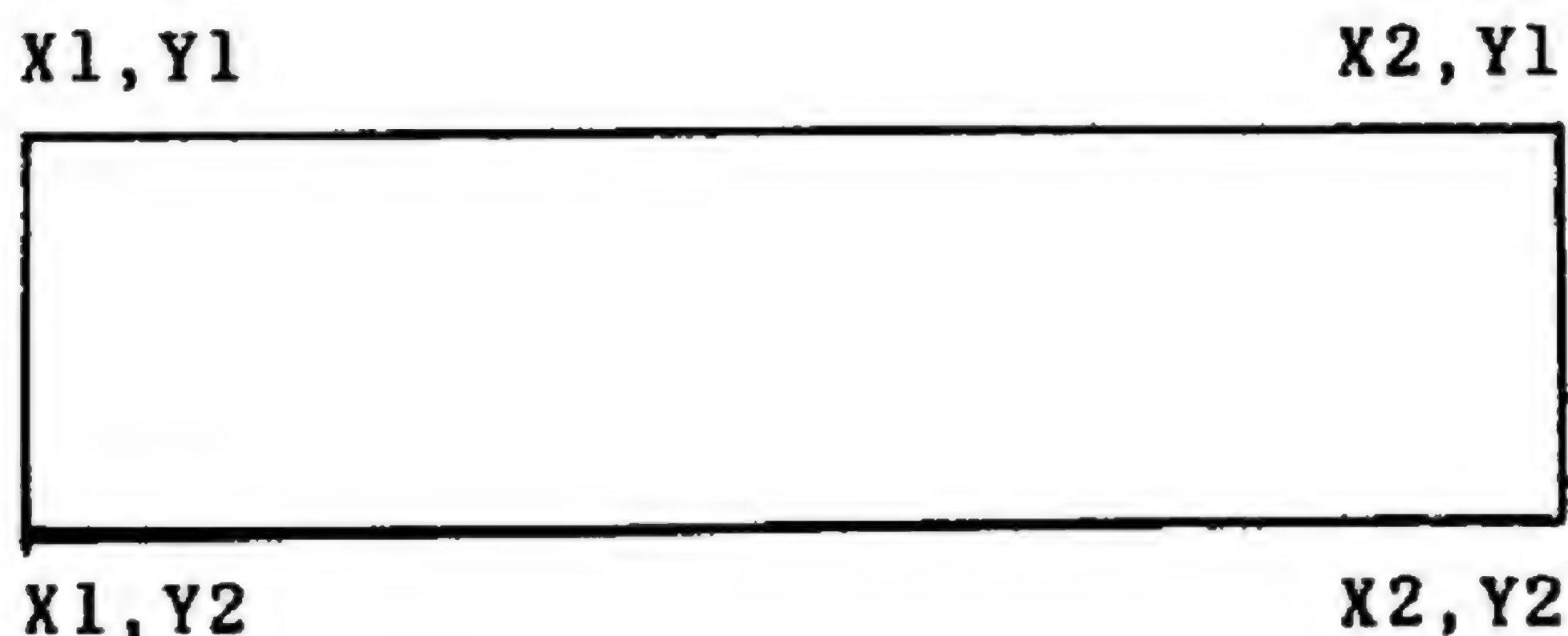
A = 3

IX + 0	=	FF	
IX + 1	=	FF	FFFF = -1
IX + 2	=	2C	
IX + 3	=	01	012C = 300
IX + 4	=	0A	
IX + 5	=	00	000A = 10

Enfin, l'étape principale consiste en l'analyse de la façon de réaliser la commande. Une programmation saine et efficace demande une bonne connaissance des routines internes de l'AMSTRAD pour éviter l'écriture de routines redondantes avec les routines système. Toutes les routines systèmes existant dans l'AMSTRAD sont décrites avec leur conditions d'entrée et de sortie dans le livre CLEFS POUR L'AMSTRAD paru aux éditions PSI.

En guise d'exemple didactique complet, nous allons réaliser la programmation d'une COMMANDE de traçage de rectangle.

Pour tracer un rectangle de façon univoque, il suffit de définir les extrémités d'une diagonale.

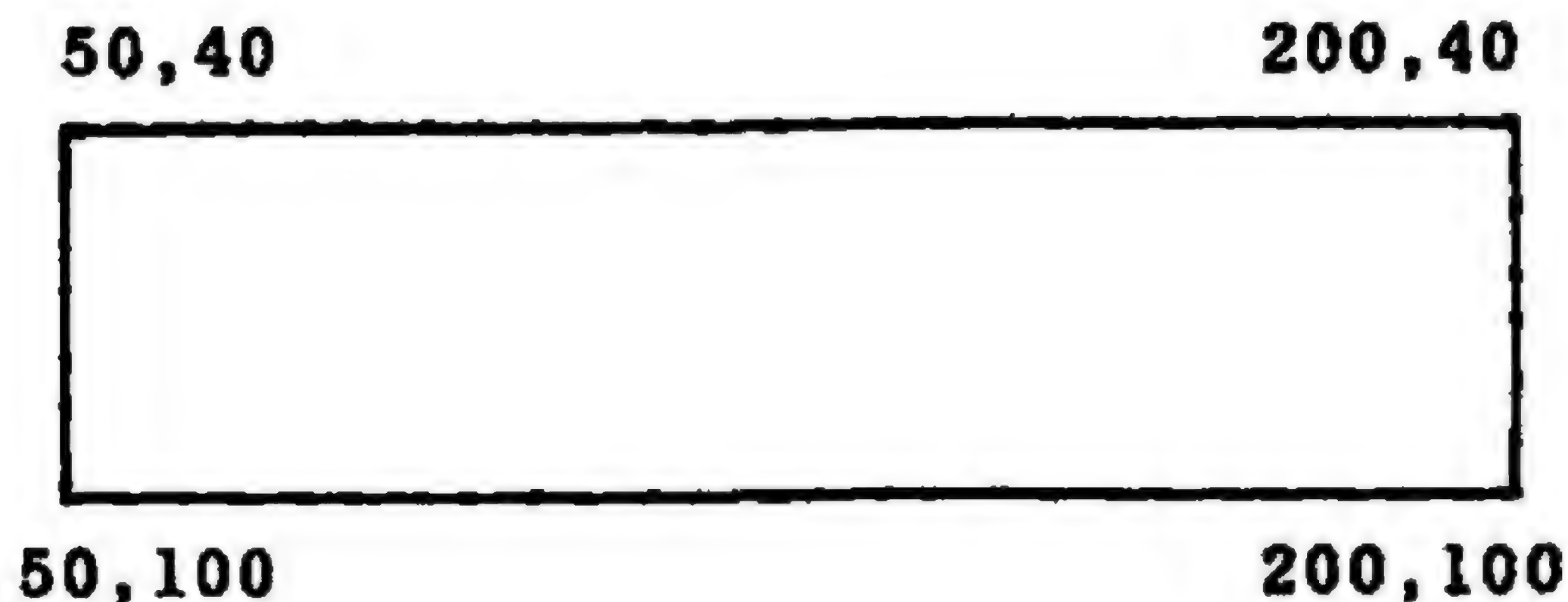


Le couple de point X1,Y1 et X2,Y2 ou le couple X2,Y1 et X1,Y2 suffisent pour définir le rectangle. En outre, il est judicieux de pouvoir définir son encre.

La syntaxe générale sera : :RECTANGLE X1,Y1,X2,Y2,ENCRE

exemple : :RECTANGLE 50,40,200,100,2

Qui produira dans l'encre 2, le rectangle



L'algorithme général peut s'écrire :

- 1 - LIRE LE PARAMETRE ENCRE**
- 2 - POSITIONNER L'ENCRE**
- 3 - LIRE LES POINTS X1,Y1,X2,Y2**
- 4 - TRACER LE POINT X1,Y1 (POINT)**
- 5 - TRACER LA DROITE X1,Y1 - X1,Y2 (LIGNE)**
- 6 - TRACER LA DROITE X1,Y2 - X2,Y2 (LIGNE)**
- 7 - TRACER LA DROITE X2,Y2 - X2,Y1 (LIGNE)**
- 8 - TRACER LA DROITE X2,Y1 - X1,Y1 (LIGNE)**

Trois routines système sont donc nécessaires : ENCRE, POINT et LIGNE. En voici la description complète.

ROUTINE : GRA SET PEN

ADRESSE : BBDE

Positionne le numéro d'encre pour le crayon.

Condition d'entrée : A contient le numéro d'encre

Condition de sortie : AF modifié

=====

ROUTINE : GRA PLOT ABSOLUE

ADRESSE : BBBA

Trace un point en coordonnées absolues.

Conditions d'entrée : DE contient l'abscisse X du point
HL contient l'ordonnée Y du point.

Conditions de sortie : AF,BC,DE et HL sont modifiés.

=====

ROUTINE : GRA LINE ABSOLUTE

ADRESSE : BBF6

Trace une ligne de la position courante du curseur graphique vers un point précisé.

Conditions d'entrée : DE contient l'abscisse X du point d'arrivée, HL contient l'ordonnée Y de ce même point.

Conditions de sortie : AF,BC,DE et HL sont modifiés.

=====

Le programme de traçage de rectangle s'écrit :

	LIGNE:	EQU	#BBF6
	POINT:	EQU	#BBBA
	ENCRE:	EQU	#BBDE
; * CHARGEMENT DE L'ENCRE DANS A			
A01D	DD 7E 00	RECTG:	LD A, (IX+0)
; * APPEL DE LA ROUTINE			
A020	CD DE BB	CALL	ENCRE
; * CHARGEMENT DE X1 DANS DE ET SAUVEGARDE DANS LA PILE			
A023	DD 5E 08	LD	E, (IX+8)
A026	DD 56 09	LD	D, (IX+9)
A029	D5	PUSH	DE
; * CHARGEMENT DE Y1 DANS DE ET SAUVEGARDE DANS LA PILE			
A02A	DD 5E 06	LD	E, (IX+6)
A02D	DD 56 07	LD	D, (IX+7)
A030	D5	PUSH	DE
; * CHARGEMENT DE X2 DANS DE ET SAUVEGARDE DANS LA PILE			
A031	DD 5E 04	LD	E, (IX+4)
A034	DD 56 05	LD	D, (IX+5)
A037	D5	PUSH	DE
; * CHARGEMENT DE Y2 DANS DE ET SAUVEGARDE DANS LA PILE			


```

A038      DD 5E 02          LD      E,(IX+2)
A03B      DD 56 03          LD      D,(IX+3)
A03E      D5                PUSH    DE
;*
;* A CET INSTANT LA PILE CONTIENT Y2,X2,Y1,X1
;*
;* CHARGEMENT DE X1 DANS DE ET Y1 DANS HL
A03F      DD 5E 08          LD      E,(IX+8)
A042      DD 56 09          LD      D,(IX+9)
A045      DD 6E 06          LD      L,(IX+6)
A048      DD 66 07          LD      H,(IX+7)
;* SAUVE X1 DANS LA PILE
A04B      D5                PUSH    DE
;*
;* A CET INSTANT LA PILE CONTIENT X1,Y2,X2,Y1,X1
;*
;* TRACAGE DU POINT X1,Y1
A04C      CD EA BB          CALL    POINT
;* RAPPEL DE X1 ET Y2 ET SAUVE Y2 DANS LA PILE
A04F      D1                POP      DE
A050      E1                POP      HL
A051      E5                PUSH    HL
;*
;* LA PILE CONTIENT Y2,X2,Y1,X1
;*
;* TRACE LA LIGNE DE X1,Y1 VERS X1,Y2
A052      CD F6 BB          CALL    LIGNE
;* RAPPEL DE X2,Y2 ET SAUVE X2
A055      E1                POP      HL
A056      D1                POP      DE
A057      D5                PUSH    DE
;*
;* LA PILE CONTIENT X2,Y1,X1
;*
;* TRACE LA LIGNE DE X1,Y2 VERS X2,Y2
A058      CD F6 BB          CALL    LIGNE
;* RAPPEL DE X2,Y1 ET SAUVE Y1
A05B      D1                POP      DE
A05C      E1                POP      HL
A05D      E5                PUSH    HL
;*
;* LA PILE CONTIENT Y1,X1
;*
;* TRACE LA LIGNE DE X2,Y2 VERS X2,Y1
A05E      CD F6 BB          CALL    LIGNE
;* RAPPEL DE X1,Y1 ET TRACAGE DE LA LIGNE FINALE
;* DE X2,Y1 VERS X1,Y1
A061      E1                POP      HL
A062      D1                POP      DE
A063      CD F6 BB          CALL    LIGNE
;*

```


;* RETOUR A L'INTERPRETEUR BASIC

;*

A066 C9

RET

REMARQUE : Ce programme vient à la suite du programme qui termine la section 8.2.

8.4 Programme BASIC de construction du RSX RECTANGLE.

Le programme assembleur de construction du RSX de la commande RECTANGLE peut être mis sous la forme d'un programme BASIC dont voici le LISTING.

```
20 MEMORY &9FFF
40 FOR i=&A000 TO &A066
50 READ A$
60 POKE i,VAL("&" + a$)
70 NEXT i
80 DATA 00,00,00,00,01,0E,A0,21,00,A0,CD,D1,BC,C9,13,A0
90 DATA C3,1D,A0,52,45,43,54,41,4E,47,4C,C5,00,DD,7E,00
100 DATA CD,DE,BB,DD,5E,08,DD,56,09,D5,DD,5E,06,DD,56,07
110 DATA D5,DD,5E,04,DD,56,05,D5,DD,5E,02,DD,56,03,D5,DD
120 DATA 5E,08,DD,56,09,DD,6E,06,DD,66,07,D5,CD,EA,BB,D1
130 DATA E1,E5,CD,F6,BB,E1,D1,D5,CD,F6,BB,D1,E1,E5,CD,F6
140 DATA BB,E1,D1,CD,F6,BB,C9
150 CALL &A004
160 NEW
```


8.5 Amélioration des RSXs.

La méthode présentée ci-dessus fonctionne parfaitement, mais elle pose un problème. Le programme écrit n'est pas indépendant de l'adresse mémoire. Il n'est pas possible de le rendre indépendant simplement, car l'adresse des tables est forcément absolue. Le code produit doit toujours être chargé au même endroit. Ce qui peut poser des problèmes si vous utilisez un système équipé de ROMs supplémentaires (CPC664 ou CPC464 avec disque). Les ROMs supplémentaires installent leurs propres RSXs et utilisent une partie de la mémoire.

Vous devez, dans la mesure du possible, essayer de rendre vos RSXs indépendants de l'adresse de chargement. Pour ce, une méthode simple consiste à écrire un programme de relocation qui précédera le programme RSX classique et de faire suivre le programme RSX par une table de toutes les adresses qui contiennent une référence absolue. Pour qu'un programme puisse disposer de sa propre adresse dans un registre afin de calculer l'OFFSET de relocation, il suffit d'utiliser la petite astuce que voici :

Le RST 30H est disponible pour l'utilisateur, il suffit d'y installer la routine de 2 octets suivante :

0030	E1	POP	HL
0031	E9	JP	(HL)

A l'endroit à identifier, il suffit de faire RST 30H, le PC se retrouve à l'adresse 30H et l'adresse qui suit l'adresse du RST se trouve dans la pile (SP). Le POP HL récupère cette adresse et le JP (HL) saute à cette adresse et effectue donc en quelque sorte le RET. Dès cet instant, vous atteignez l'adresse X avec dans HL cette adresse X.

Nous vous laissons le soin d'expérimenter cette méthode utilisée par beaucoup de programmes professionnels basés sur les RSXs.

LE CPC 664.

9.1 Introduction.

Depuis mai 1985, un nouvel AMSTRAD a fait son apparition sur le marché: le CPC 664. Pour un prix dérisoire, il incorpore un lecteur de disques en lieu et place de l'enregistreur à cassettes. Et si on ne peut que déplorer le choix du format (le 3 pouces n'est pas et ne sera jamais un standard), on ne peut que s'émerveiller devant les améliorations apportées au nouveau venu.

Il n'entre pas dans les vues de ce chapitre d'explicitier complètement la gestion disque du CPC 664 ou du 464 avec disque, ce sera le but du livre suivant de la collection (Le livre de l'amstrad tome II). Nous nous contenterons de relever les améliorations et les différences du logiciel interne du CPC 664 pour permettre une adaptation facile de tous les programmes édités à ce jour d'un modèle vers l'autre.

Le CPC 664 possède 3 ROMs de 16 K. Les deux premières contiennent le BIOS et le BASIC et sont 'semblables' à celles du 464. La troisième contient le système d'exploitation disque, elle sera analysée dans le tome II.

Pour les deux premières ROMs, nous avons dit 'semblables' et non 'identiques'. En effet, les 'géniaux' concepteurs ne se sont pas contentés de corriger certaines erreurs et d'ajouter de nouvelles fonctions. Ils ont modifiés ce que les spécialistes appellent le 'LINKAGE'. Autrement dit, à l'exception des adresses vecteurs en mémoire vive, rien ne se trouve à la même place dans le 664. Cette situation est très gênante, en particulier pour les variables internes et les routines mathématiques.

9.2 Les nouvelles instructions et fonctions du 664.

A) Les fonctions :

Si SPC et DBRR n'apportent pas vraiment un sang nouveau au 664, les 2 autres fonctions DEC\$ et COPYCHR\$ sont très précieuses.

DEC\$ permet de disposer dans une variable alphanumérique du résultat d'un PRINT USING. Cette fonction est d'une grande utilité dans les programmes de gestion et d'affichage de valeurs numériques. Exemple : A\$=DEC\$(N,"#####")

COPYCHR\$ permet de disposer dans une variable alphanumérique du caractère sous le curseur dans le stream (fenêtre) désigné. Exemple : A\$=COPYCHR\$(#1)

B : Les instructions :

Les nouvelles instructions sont très intéressantes. Pour la plupart, elles peuvent être simulées sur le CPC 464 par des CALLs, PEEKs et POKEs judicieux.

CLEAR INPUT vide le tampon d'entrée. Cette instruction est simulable sur le CPC 464 par une série de POKE dans le TAMPON clavier.

CURSOR permet d'afficher ou d'éteindre le curseur en mode système ou utilisateur. Cette instruction est simulable sur le CPC 464 par des CALLs aux adresses BB7B, BB7E, BB81 et BB84.

FILL permet le remplissage d'une surface. Pour le CPC 464, voir le RSX adéquat au chapitre 10.

FRAME permet la synchronisation de l'écriture graphique avec le retour du balayage. Pour le CPC 464, un **CALL** en **BD19** réalise la même fonction.

GRAPHICS permet le positionnement de l'encre et du crayon graphique. Pour le CPC 464, un **CALL** en **BBDE** et **BBE4** réalise la même fonction. Des **PONES** en **B338** et **B339** peuvent également convenir.

MASK permet de définir la structure d'une ligne en dessin graphique. Cette fonction est très utile pour tracer des pointillés ou des traits mixtes.

ON BREAK CONT interdit l'interruption d'un programme. Cette fonction peut être simulée par un **ON BREAK GOSUB** suivi d'un **RETURN**.

9.3 Table des variables internes.

CPC664	CPC464	FONCTION
AC01	AC1C	Sémaphore AUTO
AC02	AC1D	Ligne courante AUTO
AC04	AC1F	Incrément AUTO
AC09	AC24	WIDTH
AC0C	AC26	Adresse pour NEXT
AC12	AC2C	Adresse pour FOR
AC14	AC2E	Adresse pour WEND
AC16	AC30	11 octets pour ON....GOTO
AC8A	ACA4	Tampon d'entrée CLAVIER
AD8C	ADA6	Adresse RESUME
AD90	ADAA	Numéro d'erreur
AD91	ADAB	Adresse dernier octet exécuté
AD93	ADAD	Adresse pour STOP , END et CONT
AD98	ADB1	Numéro d'erreur pour ON ERROR
AD99	ADB2	Paramètres pour SOUND (9 octets)
ADF3	AE0C	Table des types de variables (26 octets)
AE15	AE2E	Adresse pour READ
AE17	AE30	Adresse DATA pour RESTORE
AE1B	AE34	Adresse pour ON ERROR GOTO
AE1F	AE38	Sémaphore TRON TROFF
AE55	AE72	Sauvegarde de DE pendant CALL
AE57	AE74	Sauvegarde de A pendant CALL
AE58	AE75	Sauvegarde de HL pendant CALL
AE5A	AE77	Sauvegarde de SP pendant CALL
AE5C	AE79	Adresse pour ZONE
AE5E	AE7B	Adresse pour HIMEM
AE60	AE7D	Adresse pour SYMBOL
AE64	AE81	Adresse début de programme BASIC
AE66	AE83	Adresse de fin de programme BASIC
AE68	AE85	Adresse du début de la table des variables
AE6A	AE87	Adresse début table des variables tableaux
AE6C	AE89	Adresse fin table des variables tableaux
B06F	B08C	Adresse pile BASIC
B09F	B0C1	Type accumulateur virtuel
B0A0	B0C2	Accumulateur virtuel
B113	B8F7	Mode radian/degré
B118	B800	Sémaphore inhibition message cassette
B11A	B802	Indicateur ouverture fichier
B11B	B803	Adresse du tampon de catalogue
B11D	B805	Adresse tampon de lecture
B131	B819	type de fichier cassette
B132	B81A	adresse courante tampon de lecture

B134	B81C	Adresse mémoire pour les données
B136	B81E	Longueur logique du fichier
B15F	B847	Type STREAM écriture
B162	B84A	Adresse tampon d'écriture
B176	B85F	Adresse courante tampon d'écriture
B1E5	B8CD	Caractère de synchro
B1E9	B8D1	Vitesse d'écriture
B2A6	B60A	15*16 octets ENV sonore
B396	B6FA	15*16 octets ENT sonore
B496	B34C	80 valeurs touches sans SHIFT ni CONTROL
B4E6	B39C	80 " " avec SHIFT
B536	B3EC	80 " " avec CONTROL
B586	B43C	80 " " répétition
B628	B4DE	Adresse du SCANNING clavier
B62A	B4E0	Sauvegarde temporaire du caractère
B633	B4E9	Valeur vitesse de répétition
B634	B4EA	Valeur délai avant répétition
B635	B4EB	Table de scanning
B63B	B4F1	Etat joystick 1
B63E	B4F4	Etat joystick 2
B68B	B541	Adresse table sans SHIFT ni CONTROL
B68D	B543	Adresse table avec SHIFT
B68F	B547	Adresse table avec CONTROL
B691	B549	Adresse table répétition
B693	B328	Coordonnée origine axe X
B695	B32A	Coordonnée origine axe Y
B697	B32C	Coordonnée graphique X
B699	B32E	Coordonnée graphique Y
B69B	B330	Coordonnée X bord de fenêtre graphique
B69D	B332	Coordonnée X autre bord
B69F	B334	Coordonnée Y bord de fenêtre graphique
B6A1	B336	Coordonnée Y autre bord
B6A3	B338	Encre du crayon graphique
B6A4	B339	Encre du papier graphique
B6B5	B20C	Numéro du STREAM courant
B726	B285	Position ligne curseur
B727	B286	Position colonne curseur
B728	B287	Indicateur de fenêtre
B729	B288	Ligne de début de la fenêtre courante
B72A	B289	Colonne " " " " "
B72B	B28A	Ligne de fin " " " "
B72C	B28B	Colonne " " " " "
B72E	B28D	Sémaphore inhibition curseur
B72F	B28F	Encre courante crayon
B730	B290	Encre courante papier
B731	B291	Sémaphore affichage du fond
B763	B2C3	96 octets : table des codes de contrôle
B7C2	B1C7	Octet pour l'encodage de l'encre
B7C3	B1C8	Mode écran
B7C4	B1C9	Offset de l'écran
B7C7	B1CB	OPS adresse réelle début de mémoire écran

B7D2	B1D7	Valeur 1° période de clignotement
B7D3	B1D8	Valeur 2° période de clignotement
B7D4	B1DA	32 octets avec la couleur des encres
B7F7	B1FC	1 octet pour BORDER

9.4 Table des adresses ROM.

Vous trouverez ci-dessous les principales adresses des ROMs du CPC664 avec l'adresse correspondante en 464. Pour le contenu, référez vous aux sections 6.11 et 6.12 et au CLEFS POUR L'AMSTRAD page 120 à 130.

A) ROM inférieure:

664	464	664	464	664	464	664	464
005C	005C	0099	0099	00A3	00A3	0163	0163
016A	016A	0170	0170	0176	0176	017D	017D
0183	0183	01B3	01B3	01C5	01C5	01D2	01D2
01E2	01E2	0219	021A	0227	0228	0255	0256
0276	0277	0284	0285	028D	028E	0294	0295
029A	029B	02A0	02A1	02B1	02B2	0326	0329
0330	0332	05D7	05DC	0606	060B	066F	066D
068B	068A	06F4	06F5	0728	0727	0766	0776
0776	0786	077C	0799	07A4	07BA	07B0	07C6
07D0	07E6	080B	07F2	0825	07F8	0834	0807
0848	081B	0853	0826	08BB	0888	0ABB	0AA0
0ACC	0AB1	0AE5	0ACA	0B08	0AEC	0B13	0AF7
0B33	0B3C	0B38	0B45	0B52	0B50	0B59	0B57
0B66	0B64	0BAB	0BA9	0C01	0BF9	0C0D	0C05
0C1B	0C13	0C35	0C2D	0C51	0C49	0C6D	0C68
0C70	0C6B	0C86	0C82	0C8A	0C86	0CA3	0CA0
0CE6	0CE4	0CEA	0CE8	0CEE	0CEC	0CF3	0CF1
0D16	0D14	0D1B	0D19	0DB5	0DB3	0DB9	0DB7
0DE1	0DDF	0DFC	0DFA	0E40	0E3E	0EF5	0EF3
0F26	0F49	0F8F	0FC4	0F97	102F	1070	1078
1080	1088	10E0	10E8	10FF	1107	1156	115E
1161	1169	116C	1174	1178	1180	11C6	11CE
1204	120C	124E	1256	125B	1263	1261	1268
1272	1279	127A	1281	1282	1289	1293	129A
12A2	12A9	12A7	12AE	12B6	12BD	12BC	12C3
12C2	12C9	12D0	12D3	12EE	12F1	12FA	12FD
1327	132A	1331	1334	1347	134A	1377	137A
1384	1387	13A4	13A7	13A8	13AB	13BA	13C0

664	464	664	464	664	464	664	464
13FA	1400	1406	140C	144E	144B	1455	1451
14D0	14CB	154B	1540	15A4	15B0	15D3	15DF
15F7	15F1	15FA	15F4	1602	15FC	160A	1604
1618	1612	16A1	1734	16E6	1779	1713	17A6
1729	17BC	1732	17C5	1763	17F6	166A	17FD
1771	1804	1776	180A	177C	1810	177F	1813
1782	1816	1790	1824	1793	1827	1796	182A
17A2	1836	17A5	1839	17B0	183C	193C	1945
1B5C	19E0	1B98	1A1E	1BBF	1A3C	1BC5	1A42
1BF1	1A77	1C04	1A7B	1C3C	1AB3	1C46	1ABD
1CB3	1B2E	1CDB	1B56	1CE1	1B5C	1D38	1BB3
1DB8	1C2F	1DE5	1C5C	1DF2	1C60	1DF6	1C69
1DFA	1C71	1E0B	1C82	1E19	1C90	1E2F	1CA6
1E34	1CAB	1E45	1CBD	1EC4	1D3E	1EC9	1D43
1ECE	1D48	1ED8	1D52	1EDD	1D57	1EE2	1D5C
1EEF	1D69	1FE9	1E68	2050	1ECB	206B	1EE6
2114	1F9F	21AC	204A	21CE	206C	21EB	2089
2495	2338	249A	233D	24A6	2349	24AB	234E
24BC	2370	24CE	237F	24E1	238E	288B*	2392
288B*	2528	2935	27C5	294B	27DB	2955	27E5
2958	27E8	295D	27ED	2963	27F3	296A	27FA
2970	2800	2975	2805	297D	280D	2985	2815
298D	281D	2990	2820	2996	2826	299D	282D
29A6	2836	29AF	283F	29C1	2851	2BBB	2A4B
2BBF	2A4F	2BC1	2A51	2F73	31A3	2F78	31A9
30F5	2F53	31B1	300F	31B6	3014	3225	3086
322A	308C	322F	3090	329D	30FB	32A2	3100
32A7	3105	32AC	310A	3345	31AE	3349	31B2
3353	31BC	3382	31EC	33B4	321D	33C8	3231
33D8	3241	33EE	3258	349E	333B	34A2	333F
3577	3415	3604	349E	36DF	359A	3727	35E8

* Toutes les routines cassettes sont interceptées.

B) ROM supérieure :

664	464	664	464	664	464	664	464
C033	C03F	C046	C053	C058	C090	C0D7	C0CC
C0EA	C0DF	C128	C12B	C12F	C132	C23C	C20A
C227	C212	C24B	C221	C254	C22A	C278	C24F
C283	C25A	C2A4	C262	C2A8	C276	C302	C2D2
C311	C2E1	C346	C319	C34D	C320	C42D	C3E3
C452	C417	C4E1	C48C	C509	C4B5	C532	C505
C537	C50A	C53C	C4C6	C541	C4CB	C546	C4D0
C54B	C4D5	C574	C4E9	C4EE	C579	C5D7	C529
C6A5	C5FB	C76A	C6C7	C789	C6E8	C78F	C6ED
C7B3	C70F	C7EA	C747	C81D	C776	C885	C7C3
C99A	C8E1	C9A0	C8E7	C9F8	C940	CA25	C971
CA2D	C979	CA53	C99F	CB54	CA8F	CBF4	CB23
CC29	CB5A	CC34	CB65	CC96	CBC0	CCCD	CBF8
CCD8	CC03	D11A	D0CA	D133	D0DC	D14B	D0F4
D164	D107	D16B	D10E	D1E8	D190	D242	D1EA
D246	D1EE	D26D	D219	D2AB	D256	D2B7	D25F
D2F0	D298	D2F8	D2A1	D316	D2C0	D373	D31E
D37E	D329	D3A1	D34E	D3D7	D385	D459	D409
D473	D423	D489	D439	D4DE	D494	D520	D4DB
D52C	D4E7	D530	D4EB	D534	D4EF	D539	D4F4
D563	D520	D568	D525	D56D	D52A	D572	D52F
D577	D534	D57C	D539	D581	D53E	D587	D543
D59C	D559	D5C4	D584	D653	D614	D657	D618
D65B	D61C	D691	D654	D6B9	D67D	D9F4	D9C0
DB18	DAF8	DB48	DB28	DB7F	DB77	DCCD	DCD9
DCDF	DCEB	DEC6	DDE2	DECA	DDE6	DEE5	DE01
E0C8	DFDC	E1D2	E0F7	E3AD	E2DD	E3F0	E327
E41D	E354	E451	E388	E7F3	E728	E8A3	E7DF
E9A8	E8EF	E9AC	E8F3	EA7D	E9BD	EABA	E9F6
EB02	EA3C	EB59	EAA6	ECE1	EC09	F20D	F158
F214	F15F	F21E	F16D	F228	F177	F232	F17D
F261	F1BA	F2A2	F1F6	F2A9	F1FD	F50D	F47B
F570	F4EF	F784	F69D	F8EC	F834	F8FA	F842
F964	F8BA	F969	F8C4	F98F	F8EA	F9BC	F91E
F9D3	F93C	F9D8	F943	FA07	F993	FA69	FA0A
FA6E	FA10	FA74	FA16	FA7E	FA24	FA8D	FA36
FAAD	FA57	FABE	FA77	FAB5	FAA1	FC53	FC2D
FD0C	FCCC	FD21	FCE1	FD35	FCF5	FD52	FD12
FD67	FD37	FD79	FD49	FD87	FD58	FD92	FD63
FD9C	FD6D	FDB0	FD85	FE0E	FDE8	FE13	FDED
FE86	FE8D	FEEB	FEC2	FF14	FEBC	FF2A	FF02
FF3E	FF16	FF92	FF71	FFAB	FF8A	FFFB	FFF8

9.5 Vecteurs mathématiques.

Les vecteurs mathématiques n'ont pas été conservés durant le passage au 664. Les vecteurs sur les opérations entières ont été supprimés, les autres ont changé de place.

664	464	AD 664	AD 464	FONCTION
BD5E	BD3D	2F91	2E18	COPIE (DE) -> (HL)
BD61	----	2F9F	----	CONVERSION ENTIER/FLOTTANT
BD64	BD43	2FC8	2E55	CONVERSION ENTIER/FLOTTANT
BD67	BD46	2FD9	2E66	CONVERSION FLOTTANT/ENTIER
BD6A	BD49	3001	2E8E	CONVERSION FLOTTANT/ENTIER
BD6D	BD4C	3014	2EA1	FIX
BD70	BD4F	3055	2EAC	INT
BD73	BD52	305F	2EB6	SGN
BD76	BD55	30C6	2F1D	MULTIPLIE PAD 10 EXP A
BD79	BD58	34A2	333F	ADDITION
BD7C	BDA0	3159	2FE6	RND
BD7F	BD5B	349E	3337	SOUSTRACTION
BD82	BD61	3577	3415	MULTIPLICATION
BD85	BD64	3604	349E	DIVISION
BD88	BD9D	3188	2FB7	RND
DB8B	BD6A	36DF	359A	COMPARAISON
BD8E	BD6D	3731	359A	NEGATION
BD91	BD70	3727	35E8	TESTE SIGNE
BD94	BD73	3345	31AE	POSITION RADIAN DEGRE
BD97	BD76	2F73	31A3	PI
BD9A	BD79	32AC	310A	SQRT
BD9D	BD7C	32AF	310D	PUISSANCE
BDA0	BD7F	31B6	3014	LOG
BDA3	BD82	31B1	300F	LOG10
BDA6	BD85	322F	3090	EXP
BDA9	BD88	3353	31BC	SIN
BDAC	BD8B	3349	31B2	COS
BDAF	BD8E	33C8	3231	TAN
BDB2	BD91	33D8	3241	ATN
BDB5	BD94	2FD1	2E5E	EVALUATION
BDB8	BD97	3136	2E5E	INIT rnd
BDBB	BD9A	3143	2FA1	RND

9.6 Code et adresse d'exécution des mots clé.

Les mots clé dont le code est précédé de FF (255) sont indiqués par un *. Les codes sont indiqués en hexadécimal.

MOT CLÉ	CODE	ADR	MOT CLÉ	CODE	ADR.
ABS	00*	FDB0	AFTER	80	CA25
ASC	01*	FA6E	ATN	02*	D581
AUTO	81	COEA	BIN\$	71*	F964
BORDER	82	C24B	CALL	83	F261
CAT	84	D299	CHAIN	85	EB02
CHR\$	03*	FA74	CINT	04*	FEB6
CLEAR	86	C12F	CLG	87	C509
CLOSEIN	88	D2F0	CLOSEOUT	89	D2F8
CLS	8A	C283	CONT	8B	CC96
COPYCHR\$	7E*	C29B	COS	05*	D577
CREAL	06*	FF14	CURSOR	E1	C363
DATA	8C	E9A8	DEC\$	72*	F9F8
DBF	8D	D174	DEFINT	8E	D657
DEFREAL	8F	D65B	DEFSTR	90	D653
DEG	91	D52C	DELETE	92	E7F3
DERR	49*	D12E	DI	DB	C99A
DIM	93	D6B9	DRAW	94	C53C
DRAWR	95	C541	EDIT	96	C046
EI	DC	C9A0	ELSE	97	E9B2
END	98	CC34	ENT	99	D3D7
ENV	9A	D3A1	EOF	40*	C452
ERASE	9B	D9F4	ERR	41*	D133
ERROR	9C	CB54	EVERY	9D	CA2D
EXP	07*	D563	FILL	DD	C515
FIX	08*	FE0E	FOR	9E	C5D7
FRAME	E0	BD19	FRE	09*	FC53
GOSUB	9F	C78F	GOTO	A0	C789
GRAPHICS	DE	C59D	HEX\$	73*	F969
HIMEM	42*	D14B	IF	A1	C76A
INSTR	74*	FAE5	INK	A2	C254
INKEY	0A*	D459	INKEY\$	43*	FA7E
INP	0B*	F21E	INPUT	A3	DB48
INT	0C*	FE13	JOY	0D*	D473
KEY	A4	D489	LEFT\$	75*	F9D3
LEN	0E*	FA69	LET	A5	D691
LINE	A6	DB18	LIST	A7	E1D2

MOT CLE	CODE	ADR.	MOT CLE	CODE	ADR.
LOAD	A8	EABA	LOCATE	A9	C302
LOG	0F*	D56D	LOG10	10*	D568
LOWER\$	11*	FE8C	MASK	DF	C5C3
MAX	76*	D246	MEMORY	AA	F570
MERGE	AB	EB59	MID\$	AC	FA07
MIN	77*	D242	MODE	AD	C278
MOVE	AE	C532	MOVER	AF	C537
NEXT	B0	C6A5	NEW	B1	C128
ON	B2	C885	ON BREAK	B3	C979
ON ERROR	B4	CCCD	ON SQ	B5	C9F8
OPENIN	B6	D2B7	OPENOUT	B7	D2AB
ORIGIN	B8	C4E1	OUT	B9	F228
PAPER	BA	C23C	PEEK	12*	F20D
PEN	BB	C227	PI	44*	D520
PLOT	BC	C546	PLOTR	BD	C54B
POKE	BE	F214	POS	78*	C2AD
PRINT	BF	F2A9	' (REM)	CC	E9AC
PAD	C1	D530	RANDOMIZE	C2	D59C
READ	C3	DCDF	RELEASE	C4	D373
REM	C5	E9AC	REMAIN	13*	CA53
RENUM	C6	E8A3	RESTORE	C7	DCCD
RESUME	C8	CCD8	RETURN	C9	C7B3
RIGHT\$	79*	F9D8	RND	45*	D5C4
ROUND	7A*	D26D	RUN	CA	EA7D
SAVE	CB	ECE1	SGN	14*	FF2A
SIN	15*	D572	SOUND	CC	D316
SPACE\$	16*	FAAD	SPEED	CD	D4DE
SQ	17*	D37E	SQR	18*	D534
STOP	CE	CC29	STR\$	19*	F9CB
STRING\$	7B*	FA8D	SYMBOL	CF	F784
TAG	D0	C346	TAGOFF	DL	C34D
TAN	1A*	D57C	TEST	7C*	C574
TESTR	7D*	C579	TIME	45*	D13C
TROFF	D2	DEC6	TRON	D3	DECA
UNT	1B*	FEEB	UPPER\$	1C*	F8FA
VAL	1D*	FABE	VPOS	7E*	C2A4
WAIT	D4	F2E2	WEND	D5	C81D
WHILE	D6	C7EA	WIDTH	D7	C42D
WINDOW	D8	C311	WRITE	D9	F50D
XPOS	47*	D164	YPOS	48*	D16B
ZONE	DA	F2A2	+	F4	FD0C
-	F5	FD21	*	F6	FD35
/	F7	FD52	\	F9	FD67
AND	FA	FD87	MOD	FB	FD79
OR	FC	FD92	XOR	FD	FD9C

PROGRAMMES, TRUCS ET ASTUCES.

Dans ce chapitre, vous trouverez des petits trucs et astuces qui utilisent les connaissances acquises à la lecture des chapitres précédents, autrement dit, la connaissance des périphériques, de la structure du BASIC et des instructions particulières.

Vous y trouverez aussi divers programmes plus conséquents qui vous permettront entre autres de construire automatiquement des lignes de DATA, de désassembler votre mémoire, de manipuler des vecteurs ou même de copier des cassettes.

Encoder un programme est toujours fastidieux, une seconde d'inattention est toujours possible, aussi suivez ces conseils.

- Sauvez votre programme avant de le lancer. S'il se plante, vous ne devrez pas recommencer à zéro.
- Vérifiez plusieurs fois votre travail et votre syntaxe.
- Vérifiez soigneusement vos DATA et les virgules dans ceux-ci.
- Vérifiez que vous possédez bien la configuration décrite. Dans le cas du CPC 664, faites les modifications décrites.
- !!! Lisez le mode d'emploi !!!
- Ecrivez nous chez B.C.M. si un programme refusait de 'fonctionner' après de multiples vérifications.
- Si vous désirez économiser votre énergie, une disquette reprenant tous les programmes (de plus de 5 lignes) de ce livre est disponible chez B.C.M (voir page 4).

Remarques : Quand c'est possible, le programme est présenté sous forme de RSX.

Tous les programmes en assembleur de ce livre ont été écrits à l'aide du DEVPAC AMSOFT. C'est donc la syntaxe de cet assembleur que vous retrouverez dans tous les exemples.

10.1 Formatage d'un listing

!!! Ce programme est réservé au possesseur d'une disquette.

Si vous avez essayé d'imprimer un listing sur une imprimante standard, vous avez certainement rencontré des problèmes de saut de page ou de longueur de ligne.

Le petit programme suivant devrait vous aider à formater vos listings. Il vous demandera de lui communiquer

- La taille en nombre de lignes, d'une feuille de votre papier.
- Le nombre de lignes à imprimer par page.
- le nombre de caractères à imprimer par ligne.

Pour l'utiliser, il suffit de sauver en ASCII au préalable le programme à lister au moyen de la commande SAVE"nom",A.

Tous les programmes du présent ouvrage ont été imprimés par ce petit utilitaire.

Les possesseurs de cassette pourront l'utiliser en remplaçant le canal 9 par le canal 1, mais le temps relativement long de sauvegarde sur cassette ne justifie pas son utilisation.

Programme BASIC

```
10 CPT=1
20 INPUT"NOM DU PROGRAMME ";N$
30 INPUT"TAILLE FEUILLE EN LIGNE ";LT
40 INPUT"NOMBRE DE LIGNE PAR PAGE ";LP
50 INPUT"NOMBRE DE CARACTERE PAR LIGNE ";CL
60 OPENIN N$
70 LINE INPUT #9,A$
80 IF LEN(A$)>CL THEN P$=LEFT$(A$,CL):A$=RIGHT$(A$,LEN(A$)-CL):FL=1 ELSE FL=0
90 CPT=CPT+1
95 A$=" "+A$
96 P$=" "+P$
100 IF FL=0 THEN PRINT #8,A$ ELSE PRINT #8,P$
110 IF CPT=LP THEN FOR I=1 TO LT-LP:PRINT #8,"":NEXT:CPT=1
120 IF FL=1 THEN GOTO 80
130 IF EOF THEN PRINT"TERMINE":STOP
135 P$=""
140 GOTO 70
```


10.2 Construction automatique de lignes DATA

Ce programme permet de construire automatiquement des lignes de DATA sans utiliser de routine en langage machine.

Au départ, vos lignes de DATA seront remplies d'astérisques (****). Ensuite, le programme se charge de remplacer les différents astérisques par des données, puis il disparaît en laissant uniquement les lignes de DATA en mémoire.

En commençant, ce programme vous demande l'adresse de début et l'adresse de fin de la zone mémoire à stocker dans les lignes de DATA.

La ligne 60020 fournit l'adresse de début de votre mémoire.

Le programme fouille ensuite la mémoire pour localiser l'adresse de **//. Ces signes indiquent le début de la ligne de DATA (ligne 60090). Une fois cette adresse déterminée, les données à introduire sont transformées en hexadécimal et introduites à l'intérieur de la ligne de DATA au moyen de l'instruction POKE. Enfin, la ligne est terminée par ':REM' pour que les astérisques en surnombre ne gênent pas le programme.

Lorsque tout cela est terminé, le programme constructeur s'efface pour ne laisser que les DATA.

REMARQUE: une ligne de DATA doit contenir au moins 225 astérisques et permet de stocker environ 70 valeurs. Bien entendu, vous devez prévoir suffisamment de lignes de DATA (en recopiant la ligne 50000 vers la ligne 50010 ...) pour contenir toutes vos valeurs.

EXEMPLE: si l'adresse de début de la mémoire à sauver dans les DATA vaut 1000 et l'adresse de fin 1500, il sera nécessaire de dupliquer la ligne 50000 de 10 en 10 jusqu'à la ligne 50070. Cela vous permettra de disposer de 8 lignes de DATA pouvant contenir chacune 70 valeurs, ce qui donne un total de 560 valeurs.

Programme BASIC :

```
50000 DATA **//*****
*****
*****
*****
*****
*****
60000 REM DATA PACK
60010 DEFINT A-Z
60020 INPUT"ADRESSE DE DEPART ";S
60030 INPUT"ADRESSE DE FIN ";E
60040 CLS
60050 PRINT"JE TRAVAILLE"
60060 A=&170
60080 FOR I=A TO A+3000
60090 IF PEEK(I)<>42 THEN NEXT ELSE IF PEEK(I+1)<>42 THEN NE
XT ELSE IF PEEK(I+2)<>47 THEN NEXT ELSE IF PEEK(I+3)<>47 THE
N NEXT ELSE B=I
60100 FOR I=S TO E
60110 C=PEEK(I)
60120 C$=HEX$(C,2)
60135 PRINT C$;" ";
60140 C1$=LEFT$(C$,1)
60150 C2$=RIGHT$(C$,1)
60160 C1=ASC(C1$)
60170 C2=ASC(C2$)
60180 POKE B,C1:POKE B+1,C2
60190 FL=FL+3:B=B+3:IF FL>220 THEN GOSUB 60250
60200 IF FL>0 THEN POKE B-1,44
60210 NEXT I
60220 GOSUB 60250
60230 PRINT:PRINT:PRINT"TERMINE"
60240 DELETE 60000-60290
60250 POKE B-1,32:POKE B,58:POKE B+1,197
60260 FOR J=B TO B+220
60270 IF PEEK(J)<>42 THEN GOTO 60279
60271 IF PEEK(J+1)<>42 THEN GOTO 60279
60272 IF PEEK(J+2)<>47 THEN GOTO 60279
60273 IF PEEK(J+3)<>47 THEN GOTO 60279
60274 B=J:FL=0
60275 RETURN
60279 NEXT J:FL=0:RETURN
60280 FL=0
60290 RETURN
```


10.3 SUPER DEPLACEUR

Le programme suivant est présenté sous 3 formes différentes afin d'illustrer la théorie vue dans les chapitres précédents.

- A) La forme simple avec appel par instruction CALL.
- B) La forme variable tableau avec appel par CALL.
- C) La forme RSX avec appel par l'instruction rsx DEPLACE.

Le but du programme est très simple, il demande l'adresse de départ, l'adresse d'arrivée et le nombre d'octets à transférer. Ensuite, il effectue le transfert du nombre d'octets spécifié de l'adresse de départ vers l'adresse d'arrivée.

Pour ce, l'utilisation des instructions LDIR ou LDDR du 280 sont parfaites et le programme s'écrit en quelques octets.

Le 'SUPER' de ce programme tient dans le fait qu'il détermine lui même si un LDIR ou un LDDR convient mieux en fonction des adresses et afin d'éviter un écrasement des données.

PROGRAMME 1 : Méthode du CALL simple.

Pour cette méthode, rien de spécial à dire. Notez le MEMORY à l'adresse 9FFF et l'installation du programme en A000H.

Remarquez à la ligne 40 l'expression VAL("&"+A\$) qui permet l'introduction directe des DATA en hexadécimal.

PROGRAMME BASIC CALL SIMPLE

```
10 MEMORY &9FFF
15 DEFINT a-z
20 FOR i=&A000 TO &A024
30 READ a$
40 POKE i,VAL("&" + a$)
50 NEXT i
60 DATA dd,4e,00,dd,46,01,dd,5e,02,dd,56,03,dd,6e,04,dd,66,0
5,b7,e5,ed,52,e1,38,03,ed,b0,c9,09,2b,eb,09,2b,eb,ed,b8,c9
100 INPUT"depart ";dp
110 INPUT"arrivee";ar
120 INPUT"nombre d'octets";nb
130 CALL &A000,cp,ar,nb
```

PROGRAMME 2 Méthode de la VARIABLE TABLEAU.

La méthode employée ici permet d'écrire le programme sans se soucier du MEMORY. Le programme doit bien entendu être indépendant de l'adresse d'installation.

Remarquez à la ligne 130 l'adresse de lancement donné par : @a(0).

Pour déterminer les valeurs des DATA, il suffit de prendre les octets hexadécimaux du programme précédent deux à deux et de les convertir en décimal par la fonction &H et ce en plaçant le second octet avant le premier.

Exemple : Premier couple d'octets DD 4E -> ? &H4EDD -> 20189

PROGRAMME BASIC VARIABLE TABLEAU

```
15 DEFINT a-z
20 DIM a(18)
30 FOR i=0 TO 18
40 READ v
50 a(i)=v
60 NEXT i
70 DATA 20189,-8960,326,24285,-8958,854,28381,-8956,1382,-67
29,21229,14561,-4861,-13904,11017,2539,-5333,-18195,201
100 INPUT"depart ";dp
110 INPUT"arrivee";ar
120 INPUT"nombre d'octets";nb
130 CALL @a(0),dp,ar,nb
```

PROGRAMME 3 Méthode du RSX.

Le programme est le même que celui de la première méthode. Cependant, l'ensemble des routines d'installation du RSX sont chargées et exécutées préalablement au moyen du CALL d'installation de la ligne 70.

Remarquez la ligne d'appel 130 utilisant la nouvelle instruction DEPLACE.

PROGRAMME BASIC RSX

```
10 MEMORY &9FFF
15 DEFINT a-z
20 FOR i=&A000 TO &A03F
30 READ a$
40 POKE i,VAL("&"+a$)
50 NEXT i
60 DATA 00,00,00,00,01,0e,a0,21,00,a0,cd,d1,bc,c9,13,a0,c3,1
b,a0,44,45,50,4c,41,43,c5,00,dd,4e,00,dd,46,01,dd,5e,02,dd,5
6,03,dd,6e,04,dd,66,05,b7,e5,ed,52,e1,38,03,ed,b0,c9,09,2b,e
b,09,2b,eb,ed,b8,c9
70 CALL &A000
100 INPUT"depart ";dp
110 INPUT"arrivee";ar
120 INPUT"nombre d'octets";nb
130 :DEPLACE,dp,ar,nb
```


SOURCE ASSEMBLEUR DU PROGRAMME CI-DESSUS

A000	00	00	00	00		DEFS	4	
A004	01	0E	A0		START:	LD	BC, COMXT	
A007	21	00	A0			LD	HL, #A000	
A00A	0D	D1	BC			CALL	#BCD1	
A00D	09					RET		
A00E	13	A0			COMXT:	DEFW	TABLE	
A010	03	1B	A0			JP	MOVE	
A013	44	45	50	4C	TABLE:	DEFM	'DEPLAC'	
A017	41	43						
A019	05					DEFB	'E' - #80	
A01A	00					DEFB	00	
A01B	0D	4E	00		MOVE:	LD	C, (IX+0)	BC=nombre
A01E	0D	46	01			LD	B, (IX+1)	d'octets
A021	0D	5E	02			LD	E, (IX+2)	DE=arrivée
A024	0D	56	03			LD	D, (IX+3)	
A027	0D	6E	05			LD	L, (IX+4)	HL=départ
A02A	0D	66	06			LD	H, (IX+5)	
A02D	37					OR	A	CLEAR CARRY
A02E	35					PUSH	HL	SAUVE HL
A02F	0D	52				SBC	HL, DE	HL-DE
A031	01					POP	HL	RECUPERE HL
A032	38	03				JR	C, MOVE2	SI NEGATIF
A034	3B	B0				LDIR		MOVE NORMAL
A036	09					RET		RETOUR BASIC
A037	09				MOVE2:	ADD	HL, BC	HL=HL+BC-1
A038	2B					DEC	HL	
A039	EB					EX	DE, HL	DE <--> HL
A03A	09					ADD	HL, BC	HL=HL+BC-1
A03B	2B					DEC	HL	
A03C	EB					EX	DE, HL	DE <--> HL
A03D	ED	B8				LDDR		MOVE INVERSE
A03F	09					RET		RETOUR BASIC

10.4 SUPER DUMP

Ce programme permet de réaliser le DUMP (c-à-d la copie écran ou imprimante) des deux ROMs. Ce DUMP peut être demandé en ASCII par ligne de 64 caractères avec les caractères inimprimables remplacés par des points, ou en hexadécimal par ligne de 16 octets.

Remarquez la méthode utilisée pour dérouter l'impression sur écran ou sur imprimante (ligne 320).

Remarquez aussi la petite routine en langage machine de la ligne 50 qui sélectionne la ROM demandée et qui en copie le contenu de l'adresse 6000H à 9FFFH.

Le programme pose 3 questions :

- ROM INFÉRIEURE OU SUPÉRIEURE (répondez I ou S en majuscule).
- ASCII ou HEXA (répondez A ou H en majuscule suivant votre désir).
- IMPRIMANTE OU ÉCRAN (répondez I pour obtenir un DUMP Imprimante et E pour obtenir un DUMP Écran).


```

10 MODE 2:MEMORY &5FFF:CLS
20 FOR i=&A000 TO &A00F
30 READ a$:POKE i,VAL("&" + a$)
40 NEXT i
50 DATA f3,cd,06,b9,21,00,00,11,00,60,01,ff,3f,ed,b0,c9
60 CLS:rom=-1
70 INPUT"ROM INFERIEURE OU SUPERIEURE I/S ";R$
80 R$=LEFT$(R$,1)
90 IF R$="S" THEN GOTO 130
100 IF R$="I" THEN GOTO 140
110 PRINT"REPONDEZ I OU S "
120 GOTO 70
130 rom=1:POKE &A002,0:POKE &A006,&C0
140 CALL &A000
150 INPUT"ASCII OU HEXA A/H ";R$
160 R$=LEFT$(R$,1)
170 IF R$="A" THEN TYPE=4 :GOTO 210
180 IF R$="H" THEN TYPE=1 :GOTO 210
190 PRINT"REPONDEZ A OU H "
200 GOTO 150
210 INPUT"IMPRIMANTE OU ECRAN I/E";R$
220 R$=LEFT$(R$,1)
230 IF R$="E" THEN PERIPH=0:GOTO 270
240 IF R$="I" THEN PERIPH=8:GOTO 270
250 PRINT"REPONDEZ E OU I"
260 GOTO 210
270 FOR I=&6000 TO 40959
280 IF INT(I/16/TYPE)*16*TYPE=I THEN PRINT #PERIPH,"":PRINT
#PERIPH,HEX$(I+&6000*ROM,4);" ";
290 A=PEEK(I)
300 IF TYPE=1 THEN A$=HEX$(A,2)
310 IF TYPE=4 THEN IF (A>31 AND A<127) OR A>159 THEN A$=CHR$
(A) ELSE A$="."
320 PRINT #PERIPH,A$;
330 IF TYPE =1 THEN PRINT #PERIPH," ";
340 NEXT I

```


10.5 Désassembleur en BASIC.

Il permet de désassembler la RAM.

A la première question : ADRESSE DE DEPART, vous répondez l'adresse absolue à laquelle le désassemblage doit commencer (1000 par exemple), cette adresse est en décimal par défaut, si vous désirez entrer une adresse en hexadécimal, faites-la précéder par le symbole & (exemple &A800).

La seconde question concerne l'adresse de fin de désassemblage, la même remarque que ci-dessus s'impose.

La troisième question concerne l'utilisation d'une imprimante. Si vous répondez O ou OUI à la question, le listing se déroulera sur l'imprimante sinon le listing se déroulera sur l'écran.

La quatrième question est différente suivant le choix fait à la troisième question. Si vous avez choisi l'option imprimante, on vous demande le nombre de lignes par page (il dépend de votre papier ou de votre goût). Si vous avez choisi l'option écran, on vous demande si vous voulez un arrêt en bas de page (il facilite grandement la lecture), vous répondez par OUI ou NON.

Si le listing a lieu sur écran, une cinquième question est posée, elle concerne la visualisation des caractères graphiques. IL faut savoir que le listing comporte 5 parties, la première affiche l'adresse absolue en hexadécimal, la seconde affiche le code objet en hexadécimal, la troisième affiche le code objet en ASCII, la quatrième affiche la mnémonique et la dernière les opérandes. La troisième partie est concernée par cette question car les caractères qui ne sont pas représentables dans le code ASCII sont transformés en points. Si vous répondez OUI à cette dernière question, les caractères graphiques sont affichés à l'écran en lieu et place des points de remplacement.


```

10 CLS
20 REM
30 PRINT"un moment svp"
40 GOSUB 1030
50 INPUT"adresse de depart";L
60 INPUT"adresse de fin";LF
70 P=0
80 INPUT"imprimante(o/n)";P$
90 P$=LEFT$(P$,1)
100 IF P$="O" OR P$="o" THEN GOTO 130
110 INPUT"arret en bas de page (o/n)";S$
120 S$=LEFT$(S$,1)
130 IF P$="O" OR P$="o" THEN INPUT"nombre de lignes par page
";LP
140 IF P$<>"O" AND P$<>"o" THEN INPUT"affichage des graphiqu
es (o/n)";AG$:AG$=LEFT$(AG$,1)
150 INPUT"taper enter";RP$
160 IF P$="o" OR P$="O" THEN GOSUB 190 ELSE GOSUB 290
170 IF S$="o" OR S$="O" THEN GOTO 150 ELSE GOTO 160
180 END
190 P=P+1
200 PRINT #8,CHR$(12)
210 FOR N=1 TO LP
220 GOSUB 350
230 PRINT #8,L$
240 IF L>LF THEN PRINT #8," ":PRINT #8,"TERMINE":GOTO 2450
250 NEXT N
260 PRINT #8," "
270 PRINT #8,TAB(36);"-";P;"-"
280 RETURN
290 CLS:FOR N=1 TO 22
300 GOSUB 350
310 IF L>LF THEN PRINT:PRINT"TERMINE":GOTO 2450
320 PRINT L$
330 NEXT N
340 RETURN
350 IO=PEEK(L)
360 IF IO=203 THEN 470
370 IF IO=237 THEN 520
380 IF IO=221 THEN 600
390 IF IO=253 THEN 620
400 I1=PEEK(L+1):I2=PEEK(L+2)
410 GOSUB 910
420 RN$="HL":RS$="(HL)"
430 GOSUB 800
440 GOSUB 2230
450 L$=L$+M$
460 RETURN
470 IO=PEEK(L+1)+256
480 GOSUB 910
490 IF M$="???" THEN 440

```



```

500 DL=2
510 GOTO 420
520 IO=PEEK(L+1)
530 IF IO<64 OR (IO>127 AND IO<160) OR IO>191 THEN IO=191
540 IF IO<128 THEN IO=IO+448 ELSE IO=IO+416
550 I1=PEEK(L+2):I2=PEEK(L+3)
560 GOSUB 910
570 IF M$="???" THEN 440
580 DL=DL+1
590 GOTO 420
600 RN$="IX"
610 GOTO 630
620 RN$="IY"
630 C=PEEK(L+2)
640 GOSUB 2410
650 RS$="("+RN$+"$"+C$+")"
660 IF PEEK(L+1)=203 THEN 740
670 IO=PEEK(L+1):I1=PEEK(L+2):I2=PEEK(L+3)
680 IF IO=54 THEN I1=PEEK(L+3):I2=0
690 GOSUB 910
700 FM=0:FS=0
710 IF M$<>"???" THEN GOSUB 800
720 DL=DL+(FM OR FS)+FS
730 GOTO 440
740 IO=PEEK(L+3)+256
750 GOSUB 910
760 FM=0:FS=0
770 IF M$<>"???" THEN GOSUB 800
780 DL=DL+3*FS
790 GOTO 440
800 FM=0:FS=0:I=5
810 I=I+1:IF I>LEN(M$) THEN RETURN
820 R$=MID$(M$,I,1):IF R$<>"#" AND R$<>"*" THEN 810
830 IF R$="*" THEN 880
840 FM=1
850 M$=LEFT$(M$,I-1)+RN$+RIGHT$(M$,LEN(M$)-I)
860 I=I+LEN(RN$)
870 GOTO 810
880 FS=1
890 M$=LEFT$(M$,I-1)+RS$+RIGHT$(M$,LEN(M$)-I)
900 RETURN
910 IN$=OP$(IO)
920 T=ASC(IN$)-48
930 IF T<1 OR T>9 THEN T=0 ELSE IN$=RIGHT$(IN$,LEN(IN$)-1)
940 FOR I=1 TO LEN(IN$)
950 IF MID$(IN$,I,1)=" " THEN 1000
960 NEXT I
970 IO$=IN$+STRING$(5-LEN(IN$)," ")
980 I1$=""
990 GOTO 1020
1000 IO$=LEFT$(IN$,I)+STRING$(5-I," ")

```



```

1010 I1$=RIGHT$(IN$,LEN(IN$)-I)
1020 ON T+1 GOTO 1740,1770,1820,18B0,1940,2000,2040,2100,214
0,2200
1030 DIM OP$(607),FL$(7)
1040 RESTORE
1050 FOR I=0 TO 7:READ FL$(I):NEXT I
1060 I=0
1070 READ OP$(I)
1080 IF LEFT$(OP$(I),1)<>"1" THEN 1110
1090 FOR J=1 TO 7:OP$(I+J)=OP$(I):NEXT J
1100 I=I+7
1110 I=I+1:IF I<=607 THEN 1070
1120 RETURN
1130 DATA "B","C","D","E","H","L","*","A"
1140 DATA "NOP","3LD BC","LD (BC),A","INC BC","INC B"
1150 DATA "DEC B","2LD B","RCLA","EX AF,AF',"ADD #,BC"
1160 DATA "LD A,(BC)","DEC BC","INC C","DEC C","2LD C"
1170 DATA "RRCA","4DJNZ B","3LD DE","LD (DE),A","INC DE"
1180 DATA "INC D","DEC D","2LD D","RLA","4JR","ADD #,DE"
1190 DATA "LD A,(DE)","DEC DE","INC E","DEC E","2LD E"
1200 DATA "RRA","4JR NZ","3LD #","BLD #","INC #"
1210 DATA "INC H","DEC H","2LD H","DAA","4JR Z","ADD #,#"
1220 DATA "6LD #","DEC #","INC L","DEC L","2LD L","CPL"
1230 DATA "4JR NC","3LD SP","8LD A","INC SP","INC *"
1240 DATA "DEC *","2LD *","SCF","4JR C","ADD #,SP"
1250 DATA "6LD A","DEC SP","INC A","DEC A","2LD A","CCF"
1260 DATA "1LD B","1LD C","1LD D","1LD E","1LD H","1LD L"
1270 DATA "1LD *","1LD A","1ADD A","1ADC A","1SUB A"
1280 DATA "1SBC A","1AND","1XOR","1OR","1CP"
1290 DATA "RET NZ","POP BC","3JP NZ","3JP","3CALL NZ"
1300 DATA "PUSH BC","2ADD A","RST D","RET Z","RET"
1310 DATA "3JP Z","9","3CALL Z","3CALL","2ADC A"
1320 DATA "RST $B","RET NC","POP DE","3JP NC"
1330 DATA "7OUT A","3 CALL NC","PUSH DE","2SUB A"
1340 DATA "RST $10","RET C","EXX","3JP C","SIN A"
1350 DATA "3CALL C","9","2SBC A","RST $18"
1360 DATA "RET PO","POP #","3JP PO","EX (SP),#"
1370 DATA "3CALL PO","PUSH #","2AND","RST $20"
1380 DATA "RET PE","JP (#)","3JP PE","EX DE,HL"
1390 DATA "3CALL PE","9","2XOR","RST $28"
1400 DATA "RET P","POP AF","3JP P","DI","3CALL P"
1410 DATA "PUSH AF","2OR","RST $30","RET M"
1420 DATA "LD SP,#","3JP M","EI","3CALL M"
1430 DATA "9","2CP","RST $38"
1440 REM CODE CB
1450 DATA "1RLC","1RCC","1RL","1RR"
1460 DATA "1SLA","1SRA","1DPINC","1SRL"
1470 DATA "1BIT 0","1BIT 1","1BIT 2","1BIT 3"
1480 DATA "1BIT 4","1BIT 5","1BIT 6","1BIT 7"
1490 DATA "1RES 0","1RES 1","1 RES 2","1RES 3"
1500 DATA "1RES 4","1RES 5","1RES 6","1RES 7"

```



```

1510 DATA "1SET 0","1SET 1","1SET 2","1 SET 3"
1520 DATA "1SET 4","1SET 5","1SET 6","1SET 7"
1530 REM CODE ED40-ED7F
1540 DATA "IN B,(C)","OUT (C),B","SBC HL,BC"
1550 DATA "8LD BC","NEG","RETN","IM 0","LD I,A"
1560 DATA "IN C,(C)","OUT (C),C","ADC HL,BC"
1570 DATA "6LD BC","9","RETI","9","LD R,A"
1580 DATA "IN D,(C)","OUT (C),D","SBC HL,DE"
1590 DATA "8LD DE","9","9","IM 1","LD A,I"
1600 DATA "IN E,(C)","OUT (C),E","ADC HL,DE"
1610 DATA "6LD DE","9","9","IM 2","LD A,R"
1620 DATA "IN H,(C)","OUT (C),H","SBC HL,HL"
1630 DATA "8LD HL","9","9","9","RRD"
1640 DATA "IN L,(C)","OUT (C),L","ADC HL,HL"
1650 DATA "6LD HL","9","9","9","RLD"
1660 DATA "9","9","SBC HL,SP","8LD SP"
1670 DATA "9","9","9","9","IN A,(C)","OUT (C),A"
1680 DATA "ADC HL,SP","6LD SP","9","9","9","9"
1690 REM EDAO-EDBF
1700 DATA "LDI","CPI","INI","OUTI","9","9","9","9"
1710 DATA "LDD","CPD","IND","OUTD","9","9","9","9"
1720 DATA "LDIR","CPIR","INIR","OTIR","9","9","9","9"
1730 DATA "LDDR","CPDR","INDR","OTDR","9","9","9","9"
1740 DL=1
1750 M$=I0$+I1$
1760 RETURN
1770 DL=1
1780 IF LEN(I1$)<>0 THEN I1$=I1$+","
1790 M$=I0$+I1$+FL$(IO AND 7)
1800 IF IO=118 THEN M$="HALT"
1810 RETURN
1820 DL=2
1830 IF LEN(I1$)<>0 THEN I1$=I1$+","
1840 C=I1
1850 GOSUB 2410
1860 M$=I0$+I1$+"0"+C$+"H"
1870 RETURN
1880 DL=3
1890 IF LEN(I1$)<>0 THEN I1$=I1$+","
1900 C=256*I2+I1
1910 GOSUB 2430
1920 M$=I0$+I1$+"0"+C$+"H"
1930 RETURN
1940 DL=2
1950 IF LEN(I1$)<>0 THEN I1$=I1$+","
1960 IF I1<128 THEN C=L+2+I1 ELSE C=L+2+I1-256
1970 GOSUB 2430
1980 M$=I0$+I1$+"0"+C$+"H"
1990 RETURN
2000 DL=2
2010 C=I1

```



```

2020 GOSUB 2410
2030 GOTO 2070
2040 DL=3
2050 C=256*I2+I1
2060 GOSUB 2430
2070 IF LEN(I1$)<>0 THEN I1$=I1$+", "
2080 M$=I0$+I1$+"("+"0"+C$+"H)"
2090 RETURN
2100 DL=2
2110 C=I1
2120 GOSUB 2410
2130 GOTO 2170
2140 DL=3
2150 C=256*I2+I1
2160 GOSUB 2430
2170 IF LEN(I1$)<>0 THEN I1$=", "+I1$
2180 M$=I0$+"("+"0"+C$+"H)" + I1$
2190 RETURN
2200 DL=1
2210 M$="???"
2220 RETURN
2230 C=L
2240 GOSUB 2430
2250 L$=C$+" "
2260 D$=""
2270 FOR LT=L TO L+DL-1
2280 CT=PEEK(LT)
2290 IF P$="O" OR AG$="N" THEN CT=CT AND 127
2300 IF CT>127 AND CT<160 THEN CT=CT AND 127
2310 IF CT>215 THEN CT=CT AND 127
2320 IF CT<32 OR CT=127 THEN CT=46
2330 L$=L$+CHR$(CT)
2340 C=PEEK(LT)
2350 GOSUB 2410
2360 D$=D$+C$+" "
2370 NEXT LT
2380 L$=L$+STRING$(5-DL," ")+D$+STRING$(3*(5-DL)-2," ")
2390 L=L+DL
2400 RETURN
2410 C$=HEX$(C,2)
2420 RETURN
2430 C$=HEX$(C,4)
2440 RETURN
2450 INPUT"TAPER ENTER";RP$
2460 CLS
2470 INPUT"ENCORE O/N ";RP$
2480 RP$=LEFT$(RP$,1)
2490 IF RP$="O" OR RP$="o" THEN CLS:GOTO 50
2500 IF RP$<>"N" AND RP$<>"n" THEN GOTO 2460

```


10.6 RSX sur le SCROLLING.

Le petit programme suivant introduit deux nouvelles instructions dans votre AMSTRAD. La première est appelée par la commande :SCROLLU, la seconde est appelée par la commande :SCROLLD.

:SCROLLU produit la remontée d'une ligne de l'écran. La dernière ligne devient l'avant dernière, la deuxième devient la première. La nouvelle dernière ligne devient vierge et l'ancienne première ligne est perdue. Ce procédé est appelé SCROLLING.

:SCROLLD produit l'effet contraire. La première ligne devient la deuxième, l'avant dernière devient la dernière. La nouvelle première ligne est vierge et l'ancienne dernière ligne est perdue.

Ce programme très simple utilise, en plus de la routine d'initialisation des RSXs, trois routines internes que nous vous invitons à découvrir en détails.

=====

ROUTINE : TXT GET PAPER	ADRESSE : BB99
-----	-----

Lit l'encre du papier de la fenêtre courante.

Pas de condition d'entrée.

En sortie, A contient l'encre et F est modifié.

=====

ROUTINE : SCR INK ENCODE	ADRESSE : BC2C
-----	-----

Encode un encre pour couvrir tous les points d'un octet.

Condition d'entrée : A contient le numéro d'encre.

Condition de sortie : A contient l'encre encodée
F est modifié

=====

=====

ROUTINE : SCR HW ROLL

ADRESSE : BC4D

Déplace l'écran complet d'un caractère (8 pixels)

Condition d'entrée : B=0 Déplacement vers le bas.
 B#0 Déplacement vers le haut.

Condition de sortie : AF,BC,DE et HL sont modifiés.

=====

PROGRAMME BASIC D'INITIALISATION DU RSX.

```
10 MEMORY &9FFF
20 FOR I=&A000 TO &A030
30 READ A$
40 POKE I,VAL("&" + A$)
50 NEXT I
60 CALL &A000
70 DATA 01,0A,A0,21,31,A0,CD,D1,BC,C9,12,A0,C3,21,A0,C3,2D,A
0,53,43,52,4F,4C,4C,D5,53,43,52,4F,4C,4C,C4,00,06,01,CD,99,B
B,CD,2C,BC,CD,4D,BC,C9,06,00,18,F2
```

PROGRAMME DE DEMONSTRATION.

```
10 CLS
20 PRINT"HELLO LES COPAINS"
30 PRINT"JE ME PROMENE DE 20 LIGNES"
40 PRINT"DANS LE SENS VERTICAL"
50 FOR I=1 TO 20
60 :SCROLLD
70 NEXT I
80 FOR I=1 TO 20
90 :SCROLLU
100 NEXT I
110 GOTO 50
```


Hisoft GENA3.1 Assembler. Page 1.

Pass 1 errors: 00

```

10 ; RSX SCROLLING HAUT ET BAS
20 ; D. MARTIN LIEGE 1985
30 ;
A000 40          ORG    #A000
BCD1 50 INIRSX: EQU    #BCD1
BC2C 60 INKENC: EQU    #BC2C
BC4D 70 HWROLL: EQU    #BC4D
BB99 80 GETPAP: EQU    #BB99
A000 90          LD     BC,COMEXT
A003 100         LD     HL,TAMPON
A006 110        CALL   INIRSX
A009 120        RET
A00A 130 COMEXT: DEFW   TABLE
A00C 140        JP     SCRUP
A00F 150        JP     SCRWD
A012 160 TABLE: DEFM   "SCROLL"
A018 170        DEFB   "U"+#80
A019 180        DEFM   "SCROLL"
A01F 190        DEFB   "D"+#80
A020 200        DEFB   00
A021 210 SCRUP:  LD     B,1
A023 220 NEXSCR: CALL   GETPAP
A026 230        CALL   INKENC
A029 240        CALL   HWROLL
A02C 250        RET
A02D 260 SCRWD:  LD     B,0
A02F 270        JR     NEXSCR
A031 280 TAMPON: DEFS   4
```

Pass 2 errors: 00

```

COMEXT A00A  GETPAP BB99  HWROLL BC4D
INIRSX BCD1  INKENC BC2C  NEXSCR A023
SCRWD  A02D  SCRUP  A021  TABLE A012
TAMPON A031
```

Table used: 140 from 154

10.7 ADDITION VECTORIELLE.

Dans le but d'illustrer l'utilisation des appels des vecteurs mathématiques, voici un programme qui réalise la somme de tous les éléments d'une variable tableau simple dimension (vecteur). Vous pourrez en apprécier la vitesse d'exécution, la somme de 1000 éléments s'effectuant en moins de six dixièmes de seconde.

Malheureusement, comme nous l'avons signalé, le CPC464 et le CPC664 n'ont pas leurs vecteurs mathématiques au mêmes endroits. Nous avons donc décidé de vous présenter le programme dans deux versions différentes. Vous trouverez la version CPC664 à la suite de la source assembleur du programme.

Le CPC464 utilise le vecteur d'appel situé en BD58H et le CPC664 utilise le vecteur d'appel situé en BD79H.

Le programme est composé de 2 parties. La première occupe les lignes 10 à 70 et sert à installer la routine d'addition écrite en assembleur à l'adresse A000H. La seconde est une partie de démonstration, elle occupe les lignes 100 à 190. Les lignes 100 à 155 créent 1000 valeurs aléatoires. Les lignes 158, 170, 180 et 190 servent à mesurer le temps de calcul. La ligne 160 est le coeur du programme, elle exécute l'addition proprement dite.

Remarque : le passage des paramètres est réalisé à l'aide de la fonction @ (VARPTR). Le premier paramètre est le pointeur de la variable résultat, le second paramètre est le pointeur de l'élément 0 du vecteur choisi. Le passage du pointeur de la variable résultat (@r) implique l'existence préalable de cette variable, c'est la raison d'être de la ligne 156 qui initialise la variable r à 0.

IMPORTANT : La table des variables tableaux se trouve en fin de programme BASIC. Autrement dit, dans le même espace que la routine mathématique d'addition (0000-3FFF). Le déroulement de la routine est donc complètement perturbé puisqu'elle lit les valeurs à additionner dans la ROM et non dans la RAM. L'astuce pour contourner ce problème consiste en un déplacement de la table vers le haut de mémoire. C'est le but des deux POKES aux adresses AE89 et AE8A.


```

5 REM addition vectorielle par CALL
6 REM version CPC 464
7 REM D.MARTIN LIEGE 1985
8 REM
10 MEMORY &9FFF
11 POKE &AE8A,64
12 POKE &AE89,0
20 FOR i=&A000 TO &A044
30 READ a$
40 POKE i,VAL("&" + a$)
50 NEXT i
60 DATA dd,6e,00,dd,66,01,e5,2b,46,2b,4e,03,c5,01,05,00,21,4
5,a0,e5,11,46,a0,36,00,ed,b0,e1,c1,d1,c5,d5,e5,dd,e5
70 DATA cd,58,bd,dd,e1,e1,d1,c1,0b,79,b0,28,09,e5,21,05,00,1
9,eb,e1,18,e5,dd,5e,02,dd,56,03,01,05,00,ed,b0,c9
100 REM partie demo
105 DIM a(999)
106 CIS
110 PRINT"je cree mes 1000 valeurs"
120 FOR i=0 TO 999
130 a(i)=RND(1)
140 LCCATE 2,2
150 PRINT i
155 NEXT i
156 r=0
158 depart=TIME
160 CALL &A000,@r,@a(0)
170 arrive=TIME
180 PRINT"somme = ";r
190 PRINT"temps = ";(arrive-depart)/300

```


Pass 1 errors: 00

```

10 ;addition vectorielle
20 ;D. MARTIN Crozon aout 1985
30 ;
A000 40 org #a000
50 ;* CHARGEMENT DU VARPTR DE
60 ;* L'ELEMENT 0 DU VECTEUR
A000 DD6E00 70 LD L,(IX+0)
A003 DD6601 80 LD H,(IX+1)
A006 E5 90 PUSH HL
100 ;* BC=NOMBRE ELEMENTS DU VECTEUR
A007 2B 110 DEC HL
A008 46 120 LD B,(HL)
A009 2B 130 DEC HL
A00A 4E 140 LD C,(HL)
A00B 03 150 INC BC
A00C C5 160 PUSH BC
170 ;* MISE A 0 DE LA ZONE RESULTAT
A00D 010500 180 LD BC,5
A010 2145A0 190 LD HL,TAMPON
A013 E5 200 PUSH HL
A014 1146A0 210 LD DE,TAMPON+1
A017 3600 220 LD (HL),0
A019 EDB0 230 LDIR
A01B E1 240 POP HL
A01C C1 250 POP BC
A01D D1 260 POP DE
270 ;* ADDITION ROUTINE BD58 OU BD79
A01E C5 280 NEXT: PUSH BC
A01F D5 290 PUSH DE
A020 E5 300 PUSH HL
A021 DDE5 310 PUSH IX
A023 CD58BD 320 CALL #BD58 ; * CPC 464
330 ; CALL #BD79 ; * CPC 664
A026 DDE1 340 POP IX
A028 E1 350 POP HL
A029 D1 360 POP DE
A02A C1 370 POP BC
380 ; * ELEMENT SUIVANT
A02B 0B 390 DEC BC
A02C 79 400 LD A,C
A02D B0 410 OR B
A02E 2809 420 JR Z,FINI
A030 E5 430 PUSH HL
A031 210500 440 LD HL,5
A034 19 450 ADD HL,DE
A035 EB 460 EX DE,HL
A036 E1 470 POP HL

```



```

                                475 *E
A037 18E5                      480 JR NEXT
                                490 ; * VARPTR VAR RESULTAT SUR TAMPON
A039 DD5E02                    500 FINI: LD E, (IX+2)
A03C DD5603                    510 LD D, (IX+3)
A03F 010500                    520 LD BC, 5
A042 EDB0                      530 LDIR
A044 C9                        540 RET ; * RETOUR AU
A045                          550 TAMPON: DEFS 5 BASIC

```

Pass 2 errors: 00

FINI A039 NEXT A01E TAMPON A045

PROGRAMME BASIC POUR CPC 664.

```

5 REM addition vectorielle par CALL
6 REM version CPC 664
7 REM D.MARTIN LIEGE 1985
8 REM
10 MEMORY &9FFF
11 POKE &AE6D,64
12 POKE &AE6C,0
20 FOR i=&A000 TO &A044
30 READ a$
40 POKE i,VAL("&" + a$)
50 NEXT i
60 DATA dd,6e,00,dd,66,01,e5,2b,46,2b,4e,03,c5,01,05,00,21,4
5,a0,e5,11,46,a0,36,00,ed,b0,e1,c1,d1,c5,d5,e5,dd,e5
70 DATA cd,79,bd,dd,e1,e1,d1,c1,0b,79,b0,28,09,e5,21,05,00,1
9,eb,e1,18,e5,dd,5e,02,dd,56,03,01,05,00,ed,b0,c9
100 REM partie demo
105 DIM a(999)
106 CLS
110 PRINT"je cree mes 1000 valeurs"
120 FOR i=0 TO 999
130 a(i)=RND(1)
140 LOCATE 2,2
150 PRINT i
155 NEXT i
156 r=0
158 depart=TIME
160 CALL &A000,@r,@a(0)
170 arrive=TIME
180 PRINT"somme = ";r
190 PRINT"temps = ";(arrive-depart)/300

```


10.8 RSX VECTID, VECINS et VECDEL.

Après le programme d'addition, nous vous proposons d'autres manipulations sur les vecteurs. Manipulations réalisées cette fois sous forme de RSX.

Chaque RSX est présenté individuellement. Chaque fois, le programme BASIC d'initialisation et un petit programme d'illustration vous seront proposés. A la fin de la section, vous trouverez les programmes assembleur de chacun d'entre-eux. Nous conseillons à ceux d'entre-vous qui désirent utiliser souvent ces fonctions de les regrouper ensemble dans un même programme.

A) :VECTID :

Syntaxe : :VECTID,varptr de l'élément 0 du vecteur.

Fonction : Création d'un vecteur dont tous les éléments ont la même valeur (VECTEUR IDENTITE).

Remarque : Ce RSX fonctionne aussi bien avec des vecteurs entiers, réels ou alphanumériques.

PROGRAMME BASIC D'INITIALISATION DU RSX VECTID.

```
10 MEMORY &9FFF
20 FOR I=&A000 TO &A045
30 READ A$
40 POKE I,VAL("&" + A$)
50 NEXT I
60 CALL &A000
70 DATA 00,00,00,00,01,0E,A0,21,00,A0,CD,D1,BC,C9,13,A0,C3,1
A,A0,56,45,43,54,49,C4,00,DD,6E,00,DD,66,01,E5,2B,46,2B,4E,2
B,2B,2B,2B,7E,3C,E1,E5,11,00,00,5F,19,EB,E1,F5,C5,06,00,4F,E
D,B0,C1,0B,78,B1,28,03,F1,18,F0,F1,C9
```


PROGRAMME DE DEMONSTRATION DE L'UTILISATION DU RSX VECTID.

```
10 DIM L(30)
20 L(0)=99999
30 :VECTID,@L(0)
40 FOR I=0 TO 30
50 PRINT L(I)
60 NEXT I
```

B) :VECINS

Syntaxe : :VECINS,varptr de l'élément 0,position

Fonction : Insère un élément vide à la position
spécifiée dans le vecteur spécifié.

Explication : Si on insère un élément vide en position N dans un vecteur, les éléments inférieurs à l'élément N ne sont pas modifiés, l'élément N est vide, l'ancien élément N devient l'élément N+1 et ainsi de suite. Le dernier élément est perdu. Cette fonction est très utile pour maintenir des listes triées.

PROGRAMME BASIC D'INITIALISATION DU RSX VECINS.

```
10 MEMORY &9FFF
20 FOR I=&A000 TO &A06F
30 READ A$
40 POKE I,VAL("&" + A$)
50 NEXT I
60 CALL &A000
70 DATA 00,00,00,00,01,0E,A0,21,00,A0,CD,D1,BC,C9,13,A0,C3,1
A,AC,56,45,43,49,4E,D3,00,DD,6E,02,DD,66,03,E5,2B,56,2B,5E,2
B,2E,2B,2B,7E,3C,DD,6E,00,DD,66,01,D5,E5,4F,CB,E1,CB,39,29,E
B,29,EB,CB,39,30,F8,CB,47,28,08,C1,09,EB,C1,09,EB
80 DATA 18,02,C1,C1,C1,09,EB,09,06,00,4F,2B,E5,B7,ED,42,E5,2
3,B7,ED,52,E5,C1,E1,D1,28,02,ED,B8,EB,47,3E,00,77,2B,10,FC,C
9
```


PROGRAMME DE DEMONSTRATION DE L'UTILISATION DU RSX VECINS.

```
10 DIM L(30)
20 FOR I=1 TO 30
30 L(I)=I*I
40 NEXT I
50 :VECINS,@L(0),10
60 FOR I=1 TO 30
70 PRINT I,L(I)
80 NEXT I
```

C) :VECDEL

Syntaxe : :VECDEL,varptr de l'élément 0,position

Fonction : Effacement de l'élément à la position spécifiée

Explication: L'élément à la position N est détruit, les éléments inférieurs à N ne sont pas modifiés, l'élément N+1 devient l'élément N et ainsi de suite.

PROGRAMME BASIC D'INITIALISATION DU RSX VECDEL.

```
10 MEMORY &9FFF
20 FOR I=&A000 TO &A06D
30 READ A$
40 POKE I,VAL("&"+A$)
50 NEXT I
60 CALL &A00D
70 DATA 00,0D,00,00,01,0E,A0,21,00,A0,CD,D1,BC,C9,13,A3,C3,1
A,A0,56,45,43,44,45,CC,00,DD,6E,02,DD,66,03,E5,2E,56,2B,5E,2
B,2B,2B,2B,7E,3C,DD,6E,00,DD,66,01,D5,E5,4F,CB,E1,CB,39,29,E
B,29,EB,CB,39,30,F8,CB,47,28,08,C1,09,EB,C1,09,EE
80 DATA 18,02,C1,C1,C1,09,EB,09,06,00,4F,D5,EB,09,E5,EB,B7,E
D,52,E5,C1,E1,D1,28,02,ED,B0,2B,47,3E,00,77,2B,10,FC,C9
```

PROGRAMME DE DEMONSTRATION DE L'UTILISATION DU RSX VECDEL.

```
10 DIM L(30)
20 FOR I=1 TO 30
30 L(I)=I*I
40 NEXT I
50 :VECDEL,@L(0),10
60 FOR I=1 TO 30
70 PRINT I,L(I)
80 NEXT I
```


Pass 1 errors: 00

		10 ; RSX VECTID (IDENTITE)
		20 ; D. MARTIN Liege 1985
		30 ;
BCD1		40 INIRSX: EQU #BCD1
A000		50 ORG #A000
A000		60 TAMPON: DEFS 4
A004	010EAO	70 DEBUT: LD BC, COMEXT
A007	2100AO	80 LD HL, TAMPON
A00A	CDD1BC	90 CALL INIRSX
A00D	C9	100 RET
A00E	13A0	110 COMEXT: DEFW TABLE
A010	C31AA0	120 JF VECTID
A013	56454354	130 TABLE: DEFM "VECTI"
A018	C4	140 DEFB "D"+#80
A019	00	150 DEFB 0
		160 ; charge HL avec VARPTR de l'element 0
A01A	DD6E00	170 VECTID: LD L, (IX+0)
A01D	DD6601	180 LD H, (IX+1)
A020	E5	190 PUSH HL
		200 ; charge BC avec le nombre d'element
A021	2B	210 DEC HL
A022	46	220 LD B, (HL)
A023	2B	230 DEC HL
A024	4E	240 LD C, (HL)
A025	2B	250 DEC HL
A026	2B	260 DEC HL
A027	2B	270 DEC HL
A028	2B	280 DEC HL
		290 ; charge A avec le type
A029	7E	300 LD A, (HL)
A02A	3C	310 INC A
A02B	E1	320 POP HL
A02C	E5	330 PUSH HL
A02D	110000	340 LD DE, 0
A030	5F	350 LD E, A
A031	19	360 ADD HL, DE
A032	EB	370 EX DE, HL
A033	E1	380 POP HL
		390 ; copie le 1 element dans le second
		400 ; et ainsi de suite
A034	F5	410 BOUCLE: PUSH AF
A035	C5	420 PUSH BC
A036	0600	430 LD B, 0
A038	4F	440 LD C, A
A039	EDB0	450 LDIR
		460 ; Test nombre d'element
A03B	C1	470 POP BC
A03C	0B	480 DEC BC


```

                                485 *e
A03D  78                        490      LD      A, B
A03E  B1                        500      OR      C
A03F  2803                      510      JR      Z, FINI
A041  F1                        520      POP     AF
A042  18F0                      530      JR      BOUCLE
A044  F1                        540 FINI:    POP     AF
                                550 ; retour au BASIC
A045  C9                        560      RET

```

Pass 2 errors: 03

```

BOUCLE A034  COMEXT A00E  DEBUT  A004
FINI     A044  INIRSX BCD1  TABLE A013
TAMPON  A000  VECTID A01A

```

Table used: 113 from 200

Pass 1 errors: 00

```

10 ; RSX VECINS (INSERER)
20 ; D. MARTIN Liege 1985
30 ;
BCD1 40 INIRSX: EQU #BCD1
A000 50 ORG #A000
A000 60 TAMPON: DEFS 4
A004 010EA0 70 DEBUT: LD BC, COMEXT
A007 2100A0 80 LD HL, TAMPON
A00A CDD1BC 90 CALL INIRSX
A00D C9 100 RET
A00E 13A0 110 COMEXT: DEFW TABLE
A010 C31AA0 120 JP VECINS
A013 56454349 130 TABLE: DEFM "VECIN"
A018 D3 140 DEFB "S"+#80
A019 00 150 DEFB 0
160 ; charge HL avec VARPTR element 0
A01A DD6E02 170 VECINS: LD L, (IX+2)
A01D DD6603 180 LD H, (IX+3)
A020 E5 190 PUSH HL
200 ; charge DE avec le nombre d'elements
A021 2B 210 DEC HL
A022 56 220 LD D, (HL)
A023 2B 230 DEC HL
A024 5E 240 LD E, (HL)
A025 2B 250 DEC HL
A026 2B 260 DEC HL
A027 2B 270 DEC HL
A028 2B 280 DEC HL
290 ; charge A avec le type
A029 7E 300 LD A, (HL)
A02A 3C 310 INC A
320 ; charge HL avec position insertion
A02B DD6E00 330 LD L, (IX+0)
A02E DD6601 340 LD H, (IX+1)
A031 D5 350 PUSH DE
A032 E5 360 PUSH HL
370 ; sauve le type dans C (4 bits)
A033 4F 380 LD C, A
390 ; multiplie la dimension par 2
A034 CBE1 400 SET 4, C
A036 CB39 410 SRL C
A038 29 420 BOUCLE: ADD HL, HL
A039 EB 430 EX DE, HL
A03A 29 440 ADD HL, HL
A03B EB 450 EX DE, HL
A03C CB39 460 SRL C
A03E 30F8 470 JR NC, BOUCLE
480 ; teste le type

```



```

485 *e
A040 CB47 490 BIT 0,A
A042 2808 500 JR Z,JUMP1
A044 C1 510 POP BC
A045 09 520 ADD HL,BC
A046 EB 530 EX DE,HL
A047 C1 540 POP BC
A048 09 550 ADD HL,BC
A049 EB 560 EX DE,HL
A04A 1802 570 JR JUMP2
A04C C1 580 JUMP1: POP BC
A04D C1 590 POP BC
A04E C1 600 JUMP2: POP BC
A04F 09 610 ADD HL,BC
A050 EB 620 EX DE,HL
A051 09 630 ADD HL,BC
640 ; a cet endroit, A contient le type
650 ; DE pointe sur l'element
660 ; HL pointe sur le sommet du vecteur
670 ; le STACK est vide
A052 0600 680 LD B,0
A054 4F 690 LD C,A
700 ; calcul du nombre d'octets a deplacer
A055 2B 710 DEC HL
A056 E5 720 PUSH HL
A057 B7 730 OR A
A058 ED42 740 SBC HL,BC
A05A E5 750 PUSH HL
A05B 23 760 INC HL
A05C B7 770 OR A
A05D ED52 780 SBC HL,DE
A05F E5 790 PUSH HL
A060 C1 800 POP BC
A061 E1 810 POP HL
A062 D1 820 POP DE
A063 2802 830 JR Z,JMP4
840 ; deplacement
A065 EDB8 850 LDDR
A067 EB 860 JMP4: EX DE,HL
A068 47 870 LD B,A
A069 3E00 880 LD A,0
A06B 77 890 LODP: LD (HL),A
A06C 2B 900 DEC HL
A06D 10FC 910 DJNZ LOOP
920 ; retour au BASIC
A06F C9 930 RET

```

Pass 2 errors: 00

Table used: 148 from 270

Pass 1 errors: 00

```

10 ; RSX VECDEL (DELETE)
20 ; D. MARTIN Liege 1985
30 ;
40 ; les commentaires sont identiques
50 ; a ceux de VECINS
60 ;
BCD1 70 INIRSX: EQU #BCD1
A000 80 ORG #A000
A000 90 TAMPON: DEFS 4
A004 010EAO 100 DEBUT: LD BC, COMEXT
A007 2100AO 110 LD HL, TAMPON
A00A CDD1BC 120 CALL INIRSX
A00D C9 130 RET
A00E 13AO 140 COMEXT: DEFW TABLE
A010 C31AA0 150 JP VECDEL
A013 56454344 160 TABLE: DEFM "VECDE"
A018 CC 170 DEFB "L"+#80
A019 00 180 DEFB 0
A01A DD6E02 190 VECDEL: LD L, (IX+2)
A01D DD6603 200 LD H, (IX+3)
A020 E5 210 PUSH HL
A021 2B 220 DEC HL
A022 56 230 LD D, (HL)
A023 2B 240 DEC HL
A024 5E 250 LD E, (HL)
A025 2B 260 DEC HL
A026 2B 270 DEC HL
A027 2B 280 DEC HL
A028 2B 290 DEC HL
A029 7E 300 LD A, (HL)
A02A 3C 310 INC A
A02B DD6E00 320 LD L, (IX+0)
A02E DD6601 330 LD H, (IX+1)
A031 D5 340 PUSH DE
A032 E5 350 PUSH HL
A033 4F 360 LD C, A
A034 CBE1 370 SET 4, C
A036 CB39 380 SRL C
A038 29 390 BOUCLE: ADD HL, HL
A039 EB 400 EX DE, HL
A03A 29 410 ADD HL, HL
A03B EB 420 EX DE, HL
A03C CB39 430 SRL C
A03E 30F8 440 JR NC, BOUCLE
A040 CB47 450 BIT 0, A
A042 2808 460 JR Z, JUMP1
A044 C1 470 POP BC
A045 09 480 ADD HL, BC
```


		485	*e		
A046	EB	490		EX	DE,HL
A047	C1	500		POP	BC
A048	09	510		ADD	HL,BC
A049	EB	520		EX	DE,HL
A04A	1802	530		JR	JUMP2
A04C	C1	540	JUMP1:	POP	BC
A04D	C1	550		POP	BC
A04E	C1	560	JUMP2:	POP	BC
A04F	09	570		ADD	HL,BC
A050	EB	580		EX	DE,HL
A051	09	590		ADD	HL,BC
A052	0600	600		LD	B,0
A054	4F	610		LD	C,A
A055	D5	620		PUSH	DE
A056	EB	630		EX	DE,HL
A057	09	640		ADD	HL,BC
A058	E5	650		PUSH	HL
A059	EB	660		EX	DE,HL
A05A	B7	670		OR	A
A05B	ED52	680		SBC	HL,DE
A05D	E5	690		PUSH	HL
A05E	C1	700		POP	BC
A05F	E1	710		POP	HL
A060	D1	720		POP	DE
A061	2802	730		JR	Z,RIEN
A063	EDB0	740		LDIR	
A065	2B	750	RIEN:	DEC	HL
A066	47	760		LD	B,A
A067	3E00	770		LD	A,0
A069	77	780	LOOP:	LD	(HL),A
A06A	2B	790		DEC	HL
A06B	10FC	800		DJNZ	LOOP
A06D	C9	810		RET	

Pass 2 errors: 00

BOUCLE	A038	COMEXT	A00E	DEBUT	A004
INIRSX	BCD1	JUMP1	A04C	JUMP2	A04E
LOOP	A069	RIEN	A065	TABLE	A013
TAMPON	A000	VECDEL	A01A		

Table used: 148 from 223

10.9 RSX PAINT.

Ce programme est destiné aux utilisateurs de CPC464, il permet de simuler la fonction FILL du CPC664 à l'aide d'un RSX.

La simplicité de ce programme ne lui permet pas de remplir des surfaces concaves, vous devez donc en limiter l'utilisation aux surfaces convexes. Il est évident qu'une surface concave peut toujours être décomposée en plusieurs surfaces convexes. Ainsi vous pourrez toujours colorier une surface quelconque à l'aide de plusieurs instructions :PAINT.

Pour utiliser l'instruction :PAINT, il suffit de spécifier les coordonnées X et Y d'un point interne de la surface ainsi que la couleur à utiliser.

Syntaxe : :PAINT,X,Y,couleur

Le programme utilise quatre sous-routines internes.

La routine de sélection de crayon située en BBDE déjà décrite à la page 165, la routine de traçage de points située en BBBA déjà décrite à la page 166, la routine de lecture du mode courant et la routine de test de points que nous allons vous décrire en détails.

=====

ROUTINE : SRC GET MODE

ADRESSE : BC11

Lit le mode écran courant.

Pas de condition d'entrée.

Condition de sortie : A contient le mode (0,1 ou 2).

Le FLAG ZERO est vrai si le mode est 1
il est faux dans les autres cas.

Le FLAG CARRY est vrai dans le mode 0
il est faux dans les autres cas.

=====

=====

ROUTINE : GRA TEST ABSOLUTE

ADRESSE : BBF0

Teste le point situé à la position absolue spécifiée.

Condition d'entrée : DE contient la coordonnée X du point.

HL contient la coordonnée Y du point.

Condition de sortie : A contient l'encre du point spécifié.

BC, DE et HL sont modifiés.

=====

Vous trouverez ci-dessous, le programme d'initialisation du RSX qui doit être lancé préalablement à toute utilisation de l'instruction :PAINT. A la suite de celui-ci, vous trouverez un programme très simple de démonstration de l'utilisation de l'instruction :PAINT.

PROGRAMME BASIC D'INITIALISATION DU RSX.

```
10 REM rsx PAINT
20 REM D. martin 1985
30 REM
40 MEMORY 36999
50 FOR I=&9088 TO &9186
60 READ A$
70 POKE I,VAL("&"+A$)
80 NEXT I
90 CALL 37000
100 NEW
110 DATA 01,92,90,21,86,91,CD,D1,BC,C9,97,90,C3,9C,90,50,41,
49,4E,D4,DD,7E,00,FE,10,D0,4F,DD,66,03,DD,6E,02,7C,FE,02,38,
04,7D,FE,90,D0,DD,56,05,DD,5E,04,7A,FE,03,38,04,7B,FE,80,D0,
79,F5,CD,11,BC,FE,00,20,02,0E,04,FE,01,20,02,0E,02
120 DATA FE,02,20,02,0E,01,06,00,CD,E1,BB,32,6C,91,F1,CD,DE,
BB,E5,CD,05,91,CD,2A,91,CD,54,91,28,F5,E1,CD,6E,91,20,08,CD,
05,91,CD,2A,91,18,F3,3A,6C,91,CD,DE,BB,C9,E5,D5,E5,D5,C5,CD,
F0,BB,C1,D1,E1,FE,00,20,13,E5,D5,C5,CD,EA,BB,C1,D1
130 DATA E1,EB,A7,ED,42,EB,7A,FE,FF,20,E0,D1,E1,C9,E5,D5,EB,
09,EB,7A,FE,02,20,05,7B,FE,80,30,18,E5,D5,C5,CD,F0,BB,C1,D1,
E1,FE,00,20,0B,E5,D5,C5,CD,EA,BB,C1,D1,E1,18,DB,D1,E1,C9,2B,
2B,7C,FE,FF,28,0C,E5,D5,C5,CD,F0,BB,C1,D1,E1,FE,00
140 DATA C9,3E,03,FE,04,C9,00,00,23,23,7C,FE,01,20,05,7D,FE,
8F,30,ED,E5,D5,C5,CD,F0,BB,C1,D1,E1,FE,00,C9,00
```


PROGRAMME DE DEMONSTRATION DE L'UTILISATION DU RSX.

```
10 CLS
20 PLOT 100,100
30 DRAW 300,100
40 DRAW 300,300
50 DRAW 100,300
60 DRAW 100,100
70 :PAINT,200,200,1
```

SCURCE DU PROGRAMME ASSEMBLEUR.

Les quatres pages suivantes contiennent la source du programme contenu dans le programme BASIC d'initialisation.

Le programme est conçu pour fonctionner dans les trois modes écran.

L'adresse d'organisation (départ) à été fixée en 37000 (9088H), rien ne vous empêche de fixer une autre adresse de départ si vous possédez un éditeur-assembleur.

Si vous êtes passionné par les graphiques, nous vous invitons à regrouper ce programme avec les deux suivants (RECPLEIN et CERCLE) ainsi qu'avec le programme de démonstration décrit dans le chapitre 8 pour vous constituer une mini bibliothèque de primitives graphiques utilisant la technique du RSX.

Pass 1 errors: 00

```

          10 ; RSX PAINT
          20 ; D. MARTIN LIEGE 1985
          30 ;
9088      40          org      37000
BCD1      50 INIRSX: EQU      #BCD1
BC11      60 GETMOD: EQU      #BC11
BBE1      70 GETPEN: EQU      #BBE1
BBDE      80 SETPEN: EQU      #BBDE
BBF0      90 TESTPT: EQU      #BBF0
BBEA     100 PLOT:      EQU      #BBEA
9088      110          LD      BC,COMEXT
908B      120          LD      HL,TAMPON
908E      130          CALL   INIRSX
9091      140          RET
9092      150 COMEXT: DEFW    TABLE
9094      160          JP      FILL
9097      170 TABLE: DEFM    "PAIN"
909B      180          DEFB    "T"+#80
909C      190 FILL:      LD      A,(IX+0)
909F      200          CP      16
90A1      210          RET      NC
90A2      220          LD      C,A
90A3      230          LD      H,(IX+3)
90A6      240          LD      L,(IX+2)
90A9      250          LD      A,H
90AA      260          CP      2
90AC      270          JR      C,FIL2
90AE      280          LD      A,L
90AF      290          CP      #90
90B1      300          RET      NC
90B2      310 FIL2:      LD      D,(IX+5)
90B5      320          LD      E,(IX+4)
90B8      330          LD      A,D
90B9      340          CP      3
90BB      350          JR      C,FIL3
90BD      360          LD      A,E
90BE      370          CP      #80
90C0      380          RET      NC
90C1      390 FIL3:      LD      A,C
90C2      400          PUSH    AF
90C3      410          CALL   GETMOD
90C6      420          CP      0
90C8      430          JR      NZ,FIL4
90CA      440          LD      C,4
90CC      450 FIL4:      CP      1
90CE      460          JR      NZ,FIL5
90D0      470          LD      C,2
90D2      480 FIL5:      CP      2

```


		485	*e		
90D4	2002	490		JR	NZ, FIL6
90D6	0E01	500		LD	C, 1
90D8	0600	510	FIL6:	LD	B, 0
90DA	CDE1BB	520		CALL	GETPEN
90DD	326C91	530		LD	(SAVPEN), A
90E0	F1	540		POP	AF
90E1	CDDEBB	550		CALL	SETPEN
90E4	E5	560		PUSH	HL
90E5	CD0591	570	REMP1:	CALL	SRFILL
90E8	CD2A91	580		CALL	SRFIL2
90EB	CD5491	590		CALL	SRFIL3
90EE	28F5	600		JR	Z, REMPLI
90F0	E1	610		POP	HL
90F1	CD6E91	620	NEXTFI:	CALL	SRFIL4
90F4	2008	630		JR	NZ, SKIP1
90F6	CD0591	640		CALL	SRFILL
90F9	CD2A91	650		CALL	SRFIL2
90FC	18F3	660		JR	NEXTFI
90FE	3A6C91	670	SKIP1:	LD	A, (SAVPEN)
9101	CDDEBB	680		CALL	SETPEN
9104	C9	690		RET	
9105	E5	700	SRFILL:	PUSH	HL
9106	D5	710		PUSH	DE
9107	E5	720	LOOP2:	PUSH	HL
9108	D5	730		PUSH	DE
9109	C5	740		PUSH	BC
910A	CDF0BB	750		CALL	TESTPT
910D	C1	760		POP	BC
910E	D1	770		POP	DE
910F	E1	780		POP	HL
9110	FE00	790		CP	0
9112	2013	800		JR	NZ, PASLA
9114	E5	810		PUSH	HL
9115	D5	820		PUSH	DE
9116	C5	830		PUSH	BC
9117	CDEABB	840		CALL	PLOT
911A	C1	850		POP	BC
911B	D1	860		POP	DE
911C	E1	870		POP	HL
911D	EB	880		EX	DE, HL
911E	A7	890		AND	A
911F	ED42	900		SBC	HL, BC
9121	EB	910		EX	DE, HL
9122	7A	920		LD	A, D
9123	FEFF	930		CP	#FF
9125	20E0	940		JR	NZ, LOOP2
9127	D1	950	PASLA:	POP	DE
9128	E1	960		POP	HL

		965 *e	
9129	C9	970	RET
912A	E5	980	SRFIL2: PUSH HL
912B	D5	990	PUSH DE
912C	EB	1000	LOOP4: EX DE, HL
912D	09	1010	ADD HL, BC
912E	EB	1020	EX DE, HL
912F	7A	1030	LD A, D
9130	FE02	1040	CP 2
9132	2005	1050	JR NZ, SKIP3
9134	7B	1060	LD A, E
9135	FE80	1070	CP #80
9137	3018	1080	JR NC, SKIP4
9139	E5	1090	SKIP3: PUSH HL
913A	D5	1100	PUSH DE
913B	C5	1110	PUSH BC
913C	CDF0BB	1120	CALL TESTPT
913F	C1	1130	POP BC
9140	D1	1140	POP DE
9141	E1	1150	POP HL
9142	FE00	1160	CP 0
9144	200B	1170	JR NZ, SKIP4
9146	E5	1180	PUSH HL
9147	D5	1190	PUSH DE
9148	C5	1200	PUSH BC
9149	CDEABB	1210	CALL PLOT
914C	C1	1220	POP BC
914D	D1	1230	POP DE
914E	E1	1240	POP HL
914F	18DB	1250	JR LOOP4
9151	D1	1260	SKIP4: POP DE
9152	E1	1270	POP HL
9153	C9	1280	RET
9154	2B	1290	SRFIL3: DEC HL
9155	2B	1300	DEC HL
9156	7C	1310	LD A, H
9157	FEFF	1320	CP #FF
9159	280C	1330	JR Z, SKIP5
915B	E5	1340	PUSH HL
915C	D5	1350	PUSH DE
915D	C5	1360	PUSH BC
915E	CDF0BB	1370	CALL TESTPT
9161	C1	1380	POP BC
9162	D1	1390	POP DE
9163	E1	1400	POP HL
9164	FE00	1410	CP 00
9166	C9	1420	RET

		1425	*e		
9167	3E03	1430	SKIP5:	LD	A, 3
9169	FE04	1440		CP	4
916B	C9	1450		RET	
916C		1460	SAVPEN:	DEFS	2
916E	23	1470	SRFIL4:	INC	HL
916F	23	1480		INC	HL
9170	7C	1490		LD	A, H
9171	FE01	1500		CP	1
9173	2005	1510		JR	NZ, SKIP7
9175	7D	1520		LD	A, L
9176	FE8F	1530		CP	#8F
9178	30ED	1540		JR	NC, SKIP5
917A	E5	1550	SKIP7:	PUSH	HL
917B	D5	1560		PUSH	DE
917C	C5	1570		PUSH	BC
917D	CDF0BB	1580		CALL	TESTPT
9180	C1	1590		POP	BC
9181	D1	1600		POP	DE
9182	E1	1610		POP	HL
9183	FE00	1620		CP	0
9185	C9	1630		RET	
9186		1640	TAMPON:	DEFS	4

Pass 2 errors: 00

COMEXT	9092	FIL2	90B2	FIL3	90C1
FIL4	90CC	FIL5	90D2	FIL6	90D8
FILL	909C	GETMOD	BC11	GETPEN	BBE1
INIRSX	BCD1	LOOP2	9107	LOOP4	912C
NEXTFI	90F1	PASLA	9127	PLOT	BBEA
REMP LI	90E5	SAVPEN	916C	SETPEN	BBDE
SKIP1	90FE	SKIP3	9139	SKIP4	9151
SKIP5	9167	SKIP7	917A	SRFIL2	912A
SRFIL3	9154	SRFIL4	916E	SRFILL	9105
TABLE	9097	TAMPON	9186	TESTPT	BBF0

Table used: 380 from 1000

10.10 RSX RECPLEIN.

Pour compléter la bibliothèque de primitives graphiques, nous vous proposons un RSX très simple qui réalise l'équivalent de l'instruction LINE ,BF chère au BASIC MICROSOFT.

Cette primitive dessine un rectangle et en colorie la surface interne. Cette instruction peut être simulée par le RSX RECTANGLE suivi du RSX PAINT mais l'instruction PAINT étant générale, son algorithme n'est pas optimisé pour une surface rectangulaire, son temps de traçage est donc beaucoup plus grand.

Syntaxe : :RECPLEIN,X1,Y1,X2,Y2,encore

Fonction : Trace et colorie dans l'encre spécifiée le rectangle dont le coin inférieur gauche a pour coordonnées X1,Y1 et le coin supérieur droit a pour coordonnées X2,Y2

Ce programme utilise les routines internes décrites en détails aux pages 165 et 166.

Le coloriage est réalisé par le traçage de Y2-Y1 lignes horizontales comprises entre X1 et X2.

Le programme BASIC d'initialisation du RSX est suivi d'un petit programme de démonstration et de la source assembleur de la routine.

PROGRAMME BASIC D'INITIALISATION DU RSX.

```
10 MEMORY &9FFF
20 FOR I=&A000 TO &A06D
30 READ A$
40 POKE I,VAL("&" + A$)
50 NEXT
60 CALL &A000
70 DATA 01,0A,A0,21,71,A0,CD,D1,BC,C9,0F,A0,C3,13,A0,52,45,4
3,50,4C,45,49,CE,00,DD,7E,00,CD,DE,BB,DD,6E,02,D3,66,03,DD,5
E,06,DD,56,07,A7,ED,52,DA,6C,A0,22,6D,A0,DD,5E,03,DD,56,09,D
D,6E,06,DD,66,07,E5,D5,CD,EA,BB,DD,5E,04,DD,56,05
80 DATA DD,E1,FD,E1,D5,FD,E5,E1,CD,F6,BB,DD,E5,D1,DD,E1,FD,2
3,2A,6D,A0,D5,11,01,00,A7,ED,52,22,6D,A0,D1,20,E2,C9,00,00,0
0
```

PROGRAMME DE DEMONSTRATION.

```
10 CLS
20 :RECPLEIN,100,100,300,300,1
30 :RECPLEIN,150,150,350,350,2
40 :RECPLEIN,200,200,400,400,3
50 :RECPLEIN,125,125,275,275,0
```


Pass 1 errors: 00

A000		10	ORG	#A000
BCD1		20	INIRSX:	EQU #BCD1
BBDE		30	ENCRE:	EQU #BBDE
BBF6		40	LIGNE:	EQU #BBF6
BBEA		50	POINT:	EQU #BBEA
		60	; INITIALISATION RSX	
A000	010AA0	70	DEBUT:	LD BC,COMEXT
A003	2171A0	80		LD HL,TAMPON
A006	CDD1BC	90		CALL INIRSX
A009	C9	100		RET
A00A	0FA0	110	COMEXT:	DEFW TABLE
A00C	C318A0	120		JP RECP LN
A00F	52454350	130	TABLE:	DEFM "RECPLEI"
A016	CE	140		DEFB "N"+#80
A017	00	150		DEFB 00
		160	; SELECTION DE L'ENCRE	
A018	DD7E00	170	RECP LN:	LD A,(IX+0)
A01B	CDDEBB	180		CALL ENCRE
		190	; CALCUL NOMBRE DE LIGNES	
A01E	DD6E02	200		LD L,(IX+2)
A021	DD6603	210		LD H,(IX+3)
A024	DD5E06	220		LD E,(IX+6)
A027	DD5607	230		LD D,(IX+7)
A02A	A7	240		AND A
A02B	ED52	250		SBC HL,DE
A02D	DA6CA0	260		JP C,FIN
		270	; SAUVE COMPTEUR LIGNE	
A030	226DA0	280		LD (CPTR),HL
		290	; CHARGE POINT DEPART	
A033	DD5E08	300		LD E,(IX+8)
A036	DD5609	310		LD D,(IX+9)
A039	DD6E06	320		LD L,(IX+6)
A03C	DD6607	330		LD H,(IX+7)
A03F	E5	340		PUSH HL
A040	D5	350		PUSH DE
A041	CDEABB	360		CALL POINT
A044	DD5E04	370		LD E,(IX+4)
A047	DD5605	380		LD D,(IX+5)
A04A	DDE1	390		POP IX
A04C	FDE1	400		POP IY
		410	; BOUCLE PRINCIPALE	
		420	; TRACAGE DES LIGNES	
A04E	D5	430	BOUC:	PUSH DE
A04F	FDE5	440		PUSH IY
A051	E1	450		POP HL


```

                                455 *e
A052 CDF6BB 460 CALL LIGNE
A055 DDE5 470 PUSH IX
A057 D1 480 POP DE
A058 DDE1 490 POP IX
A05A FD23 500 INC IY
A05C 2A6DA0 510 LD HL,(CPTR)
A05F D5 520 PUSH DE
A060 110100 530 LD DE,1
A063 A7 540 AND A
A064 ED52 550 SBC HL,DE
A066 226DA0 560 LD (CPTR),HL
A069 D1 570 POP DE
A06A 20E2 580 JR NZ,BOUC
A06C C9 590 FIN: RET
A06D 600 CPTR: DEFS 4
A071 610 TAMPON: DEFS 4
```

Pass 2 errors: 00

BOUC	A04E	COMEXT	A00A	CPTR	A06D
DEBUT	A000	ENCRE	BBDE	FIN	A06C
INIRSX	BCD1	LIGNE	BBF6	POINT	BBEA
RECPLN	A018	TABLE	A00F	TAMPON	A071

Table used: 157 from 210

10.11 RSX CERCLE.

Pour terminer les RSX graphiques, nous vous proposons d'analyser en détails la programmation de l'instruction de traçage de cercle. Cette instruction, présente dans le BASIC de nombreux micros, fait cruellement défaut sur l'AMSTRAD.

Syntaxe : ;CERCLE,X,Y,Rayon,Couleur

X et Y sont les coordonnées du centre du cercle.

Le programme assembleur utilise six routines internes. La routine d'initialisation du RSX (BCD1), la routine de sélection de l'encre (BBDE) et la routine de traçage d'un point (BBEA) ont déjà été analysées en détails dans le chapitre 8. Voici la description des trois autres.

=====

ROUTINE : GRA SET ORIGIN

ADRESSE : BBC9

Positionne le point origine à un endroit choisi.

Condition d'entrée : DE contient la coordonnée X.

HL contient la coordonnée Y.

Condition de sortie : AF, BC, DE et HL sont modifiés.

=====

ROUTINE : GRA GET ORIGIN

ADRESSE : BBCC

Lit l'origine courante.

Pas de condition d'entrée.

Condition de sortie : DE contient la coordonnée X.

HL contient la coordonnée Y.

=====

=====

ROUTINE : TXT OUTPUT

ADRESSE : BB5A

Affiche à l'écran le caractère contenu dans l'accumulateur en respectant les codes de contrôle.

Condition d'entrée : A contient le caractère à écrire
Condition de sortie : Tout est préservé.

=====

Avant d'aborder l'analyse du problème, signalons que dans ce programme, nous avons ajouté une routine de test du nombre de paramètres (ici 4). Si le nombre de paramètres est incorrect, un message d'erreur est affiché à l'écran. Nous vous invitons à introduire cette routine dans les autres exemples de RSX de ce livre.

ANALYSE DU PROBLEME.

Pour tracer un cercle, il y a deux méthodes habituelles.

La première utilise l'équation polaire du cercle et fait appel aux fonctions sinus et cosinus. Le programme BASIC correspondant peut s'écrire :

```
10 CLS
20 R=100 : Y=200 : X=300
30 PLOT X+R,Y
40 FOR I=0 TO 2*PI STEP .1
50 X1=X+R*COS(I)
30 Y1=Y+R*SIN(I)
40 DRAW X1,Y1
50 NEXT I
```

Où R est le rayon et X,Y les coordonnées du centre du cercle.

Cette méthode est difficile à programmer en assembleur et elle demande le calcul de nombreux sinus et cosinus (62 dans l'exemple précédent).

La seconde utilise le théorème de Pythagore :

Dans un triangle rectangle, Le carré de l'hypothénuse est égal à la somme des carrés des deux autres côtés.

Si X est un côté du triangle, Y l'autre côté et R l'hypothénuse (RAYON) il vient :

$$Y^2 = R^2 - X^2$$

Pour tracer un quart de cercle, il suffit de calculer toutes les valeurs de Y pour X variant de 0 à R.

Pour tracer les trois autres quarts, il suffit de tracer les points miroirs (-X,Y) (X,-Y) (-X,-Y).

Le programme BASIC équivalent s'écrit :

```
10 CLS
20 X=300 : Y=200 : R=100
30 FOR I=0 TO R
40 Y1=SQR(R*R-I*I)
50 PLOT X+I,Y+Y1
60 PLOT X-I,Y+Y1
70 PLOT X-I,Y-Y1
80 PLOT X+I,Y-Y1
90 NEXT I
```

Cette méthode, simple dans son principe, présente quelques inconvénients. Quand X approche de R, la qualité de l'arc de cercle laisse à désirer. De plus, le calcul d'une racine carrée est gourmand en temps processeur.

Une variation de la méthode précédente basé sur l'algorithme de BRESENHAM va nous donner la solution de notre problème.

Les points seront calculés avec X variant de 0 à PI/4 à la place de 0 à R. Cette méthode produit également un quart de cercle et nous utiliserons le système de l'image miroir décrit ci-dessus pour les trois autres quarts.

Le coeur de l'algorithme est une routine qui choisit le point réel (coordonnée entière) le plus proche du point du cercle parfait. La distance entre le point choisi et le point parfait est appelée E (erreur).

Si le point est tracé en X,Y l'erreur est obtenue par :

$$E = (X^2 + Y^2) - R^2$$

En minimisant l'erreur pour chaque point, la meilleure approximation du cercle est obtenue.

Remarque : Si A est le point qui vient d'être tracé avec comme coordonnées X_1, Y_1 : alors le point suivant du cercle aura comme coordonnées X_1+1, Y_1 ou X_1, Y_1+1 . Appelons le premier point B et le second C.

$$EB = (X_B^2 + Y_B^2) - R^2 \quad \text{et} \quad EC = (X_C^2 + Y_C^2) - R^2$$

Si EB est inférieur à EC alors B est tracé sinon C est tracé.

On appelle ET l'erreur totale formée de la somme de EB et EC.

Un long développement mathématique permet de déterminer que si ET est l'erreur totale au point P alors l'erreur totale au point suivant (ET+1) sera donnée par la formule :

$$ET+1 = ET + 4X + 6 \quad \text{si le point choisi est B.}$$

ou

$$ET+1 = ET + 4(X-Y) + 10 \quad \text{si le point choisi est C.}$$

Le problème se ramène à deux additions et 2 SHIFTS (multiplication par 4) dans le premier cas et à deux additions, une soustraction et deux SHIFTS dans le second cas. Cette solution est nettement plus rapide pour un processeur que l'extraction d'une racine ou le calcul d'un sinus.

Vous noterez la multiplication par 4 aux lignes 710 à 740 et 840 à 870 du programme assembleur ainsi que l'addition de la constante 6 aux lignes 750 et 760 ou de la constante 10 aux lignes 880 et 890 du même programme.

PROGRAMME BASIC D'INITIALISATION DU RSK.

```

10 MEMORY &9FFF
20 FOR i=&A000 TO &A152
30 READ a$
40 POKE i,VAL("&"+a$)
50 NEXT i
60 CALL &A000
70 DATA 01,0A,A0,21,5F,A1,CD,D1,BC,C9,0F,A0,C3,16,A0,43,45,5
2,43,4C,C5,00,FE,04,C2,29,A1,CD,CC,BB,ED,53,53,A1,22,55,A1,D
D,7E,00,CD,DE,BB,DD,56,07,DD,5E,06,DD,66,05,DD,6E,04,E5,DD,6
6,03,DD,6E,02,22,5D,A1,E1,CD,C9,BB,01,00,00,ED,43
80 DATA 57,A1,2A,5D,A1,22,59,A1,CB,25,CB,14,E5,D1,21,03,00,A
F,ED,52,22,5B,A1,2A,57,A1,ED,5B,59,A1,E5,D5,CD,C9,A0,D1,E1,A
F,ED,52,F2,1E,A1,2A,5B,A1,01,00,00,ED,42,F2,98,A0,ED,5B,57,A
1,CB,23,CB,12,CB,23,CB,12,21,06,00,19,ED,5B,5B,A1
90 DATA 19,C3,BC,A0,2A,57,A1,ED,5B,59,A1,AF,ED,52,CB,25,CB,1
4,CB,25,CB,14,11,0A,00,19,ED,5B,5B,A1,19,ED,5B,59,A1,1B,ED,5
3,59,A1,22,5B,A1,2A,57,A1,23,22,57,A1,C3,61,A0,ED,5B,57,A1,2
A,59,A1,CD,EA,BB,ED,5B,59,A1,2A,57,A1,CD,EA,BB,21
100 DATA 00,00,ED,4B,57,A1,AF,ED,42,E5,E5,ED,5B,59,A1,CD,EA,
BB,D1,2A,59,A1,CD,EA,BB,21,00,00,ED,4B,59,A1,AF,ED,42,E5,E5,
ED,5B,57,A1,CD,EA,BB,D1,2A,57,A1,CD,EA,BB,E1,D1,E5,D5,CD,EA,
BB,E1,D1,CD,EA,BB,C9,ED,5B,53,A1,2A,55,A1,CD,C9,BB
110 DATA C9,21,36,A1,7E,FE,00,C8,CD,5A,BB,23,18,F6,4E,4F,4D,
42,52,45,20,44,27,41,52,47,55,4D,45,4E,54,53,20,49,4E,43,4F,
52,52,45,43,54,00

```

PROGRAMME DE DEMONSTRATION.

```

10 REM L'oeil du maitre
20 REM D. MARTIN Septembre 1985
30 CLS
40 FOR i=0 TO 2*PI STEP 0.1
50 :CERCLE,320+285*COS(i),200+165*SIN(i),30,1
60 NEXT
70 FOR i=1 TO 10
80 :CERCLE,320,200,i*10,2
90 NEXT

```


Pass 1 errors: 00

A000		10	ORG	#A000
BCD1		20	INIRSX:	EQU #BCD1
BBDE		30	ENCRE:	EQU #BBDE
BBC9		40	SETORG:	EQU #BBC9
BBCC		50	GETORG:	EQU #BBCC
BBEA		60	POINT:	EQU #BBEA
BB5A		70	PRINT:	EQU #BB5A
		80	; INITIALISATION RSX	
A000	010AA0	90	DEBUT:	LD BC, COMEXT
A003	215FA1	100		LD HL, TAMPON
A006	CDD1BC	110		CALL INIRSX
A009	C9	120		RET
A00A	0FA0	130	COMEXT:	DEFW TABLE
A00C	C316A0	140		JP CERCLE
A00F	43455243	150	TABLE:	DEFM "CERCL"
A014	C5	160		DEFB "E"+#80
A015	00	170		DEFB 00
		180	; TEST NOMBRE ARGUMENTS	
A016	FE04	190	CERCLE:	CP 4
A018	C229A1	200		JP NZ, ERREUR
		210	; LECTURE ET SAUVEGARDE	
		220	; TEMPORAIRE DE L'ORIGINE	
A01B	CDCCBB	230		CALL GETORG
A01E	ED5353A1	240		LD (XORG), DE
A022	2255A1	250		LD (YORG), HL
		260	; LECTURE ENCRE	
A025	DD7E00	270		LD A, (IX+0)
A028	CDDEBB	280		CALL ENCRE
		290	; LECTURE DU CENTRE	
A02B	DD5607	300		LD D, (IX+7)
A02E	DD5E06	310		LD E, (IX+6)
A031	DD6605	320		LD H, (IX+5)
A034	DD6E04	330		LD L, (IX+4)
A037	E5	340		PUSH HL
		350	; LECTURE DU RAYON	
A038	DD6603	360		LD H, (IX+3)
A03B	DD6E02	370		LD L, (IX+2)
A03E	225DA1	380		LD (RAYON), HL
A041	E1	390		POP HL
		400	; NOUVELLE ORIGINE AU CENTRE	
A042	CDC9BB	410		CALL SETORG
A045	010000	420		LD BC, 0
A048	ED4357A1	430		LD (X), BC
A04C	2A5DA1	440		LD HL, (RAYON)
A04F	2259A1	450		LD (Y), HL

		455	*E		
A052	CB25	460		SLA	L
A054	CB14	470		RL	H
A056	E5	480		PUSH	HL
A057	D1	490		POP	DE
A058	210300	500		LD	HL, 3
A05B	AF	510		XOR	A
A05C	ED52	520		SBC	HL, DE
A05E	225BA1	530		LD	(CALCUL), HL
		540	; CALCUL DES	POINTS	
		550	; VOIR TEXTE		
A061	2A57A1	560	CALC:	LD	HL, (X)
A064	ED5B59A1	570		LD	DE, (Y)
A068	E5	580		PUSH	HL
A069	D5	590		PUSH	DE
A06A	CDC9A0	600		CALL	IMAGE
A06D	D1	610		POP	DE
A06E	E1	620		POP	HL
A06F	AF	630		XOR	A
A070	ED52	640		SBC	HL, DE
A072	F21EA1	650		JP	P, FIN
A075	2A5BA1	660		LD	HL, (CALCUL)
A078	010000	670		LD	BC, 0
A07B	ED42	680		SBC	HL, BC
A07D	F298A0	690		JP	P, INFER
A080	ED5B57A1	700		LD	DE, (X)
A084	CB23	710		SLA	E
A086	CB12	720		RL	D
A088	CB23	730		SLA	E
A08A	CB12	740		RL	D
A08C	21060C	750		LD	HL, 6
A08F	19	760		ADD	HL, DE
A090	ED5B5FA1	770		LD	DE, (CALCUL)
A094	19	780		ADD	HL, DE
A095	C3BCAC	790		JP	SUITE
A098	2A57A1	800	INFER:	LD	HL, (X)
A09B	ED5B59A1	810		LD	DE, (Y)
A09F	AF	820		XOR	A
A0A0	ED52	830		SBC	HL, DE
A0A2	CB25	840		SLA	L
A0A4	CB14	850		RL	H
A0A6	CB25	860		SLA	L
A0A8	CB14	870		RL	H
A0AA	110A00	880		LD	DE, 10
A0AD	19	890		ADD	HL, DE
A0AE	ED5B5BA1	900		LD	DE, (CALCUL)

		905	*E		
A0B2	19	910		ADD	HL,DE
A0B3	ED5B59A1	920		LD	DE,(Y)
A0B7	1B	930		DEC	DE
A0B8	ED5359A1	940		LD	(Y),DE
A0BC	225BA1	950	SUITE:	LD	(CALCUL),HL
A0BF	2A57A1	960		LD	HL,(X)
A0C2	23	970		INC	HL
A0C3	2257A1	980		LD	(X),HL
A0C6	C361A0	990		JP	CALC
		1000	; TRACE DES POINTS		
		1010	; DANS LES 4 QUADRAWTS		
A0C9	ED5B57A1	1020	INAGE:	LD	DE,(X)
A0CD	2A59A1	1030		LD	HL,(Y)
A0D0	CDEABB	1040		CALL	POINT
A0D3	ED5B59A1	1050		LD	DE,(Y)
A0D7	2A57A1	1060		LD	HL,(X)
A0DA	CDEABB	1070		CALL	POINT
A0DD	210000	1080		LD	HL,0
A0E0	ED4B57A1	1090		LD	BC,(X)
A0E4	AF	1100		XOR	A
A0E5	ED42	1110		SBC	HL,bc
A0E7	E5	1120		PUSH	HL
A0E8	E5	1130		PUSH	HL
A0E9	ED5B59A1	1140		LD	DE,(Y)
A0ED	CDEABB	1150		CALL	POINT
A0F0	D1	1160		POP	DE
A0F1	2A59A1	1170		LD	HL,(Y)
A0F4	CDEABB	1180		CALL	POINT
A0F7	210000	1190		LD	HL,0
A0FA	ED4B59A1	1200		LD	BC,(Y)
A0FE	AF	1210		XOR	A
A0FF	ED42	1220		SBC	HL,BC
A101	E5	1230		PUSH	HL
A102	E5	1240		PUSH	HL
A103	ED5B57A1	1250		LD	DE,(X)
A107	CDEABB	1260		CALL	POINT
A10A	D1	1270		POP	DE
A10B	2A57A1	1280		LD	HL,(X)
A10E	CDEABB	1290		CALL	POINT
A111	E1	1300		POP	HL
A112	D1	1310		POP	DE
A113	E5	1320		PUSH	HL
A114	D5	1330		PUSH	DE
A115	CDEABB	1340		CALL	POINT
A118	E1	1350		POP	HL


```

1355 *E
A119 D1 1360 POP DE
A11A CDEABB 1370 CALL POINT
A11D C9 1380 RET
1390 ; REMISE DE L'ORIGINE
A11E ED5B53A1 1400 FIN: LD DE,(XORG)
A122 2A55A1 1410 LD HL,(YORG)
A125 CDC9BB 1420 CALL SETORG
A128 C9 1430 RET
1440 ; IMPRESSION DU MESSAGE
1450 ; EN CAS D'ERREUR
A129 2136A1 1460 ERREUR: LD HL,MESERR
A12C 7E 1470 ENVOI: LD A,(HL)
A12D FE00 1480 CP 00
A12F C8 1490 RET Z
A130 CD5ABB 1500 CALL PRINT
A133 23 1510 INC HL
A134 18F6 1520 JR ENVOI
A136 4E4F4D42 1530 MESERR: DEFM "NOMBRE D'ARGUMENTS"
A148 20494E43 1540 DEFM " INCORRECT"
A152 00 1550 DEFB 0
A153 1560 XORG: DEFS 2
A155 1570 YORG: DEFS 2
A157 1580 X: DEFS 2
A159 1590 Y: DEFS 2
A15B 1600 CALCUL DEFS 2
A15D 1610 RAYON: DEFS 2
A15F 1620 TAMPON DEFS 4
```

Pass 2 errors: 00

CALC	A061	CALCUL	A15B	CERCLE	A016
COMEXT	A00A	DEBUT	A000	ENCRE	BBDE
ENVOI	A12C	ERREUR	A129	FIN	A11E
GETORG	BBCC	IMAGE	A0C9	INFER	A098
INIRSX	BCD1	MESERR	A136	POINT	BBEA
PRINT	BB5A	RAYON	A15D	SETORG	BBC9
SUITE	A0BC	TABLE	A00F	TAMPON	A15F
X	A157	XORG	A153	Y	A159
YORG	A155				

Table used: 309 from 374

Le système de sauvegarde sur cassette permet l'écriture de fichier par blocs de 2 K à des vitesses de 1000 et 2000 bauds.

Le programme que nous allons détailler ci-dessous ajoute deux RSX, SUPERSAUE et SUPERCHARGE, qui permettent la sauvegarde de blocs de différentes longueurs à partir de n'importe quelle adresse mémoire.

Il est à noter que l'étude de ce programme demande une bonne compréhension des différentes notions abordées dans le présent ouvrage. Il nécessite notamment la connaissance des RSX, du passage des paramètres de la structure des variables, du principe de lecture et d'écriture de fichiers sur cassette, du gestionnaire d'écran et du gestionnaire cassette.

La commande SUPERSAUE peut être formulée de deux façons différentes:

1- La commande suivie de nom de fichier:

```
10 SAV="SAUE": :SUPERSAUE,@SAV
```

Dans ce cas, le système sauve par défaut un bloc de 16 K à partir de l'adresse #C000, c'est à dire la mémoire écran.

2- La commande suivie du nom de fichier, adresse départ, longueur DATA:

```
10 SAV="SAUE": :SUPERSAUE,@SAV,25000,3000
```

Dans ce cas, il sauve 3000 octets à partir de l'adresse 25000.

La commande SUPERCHARGE, quant à elle, demande un ou deux paramètres:

1- 10 :SUPERCHARGE

Chargera en mémoire le premier programme rencontré.

2- 10 LECT="SAUVE": : SUPERCHARGE, @LECT

Chargera uniquement le fichier SAUVE.

3- 10 LECT="SAUVE": : SUPERCHARGE, @LECT, N

Chargera le fichier SAUVE sans qu'aucun message ne soit affiché sur l'écran. Ceci est très pratique lorsqu'un chargement de la mémoire écran est demandé.

Lors de la sauvegarde, un Header de 30 octets est écrit devant le bloc de Data. Il est constitué du nom du fichier à sauver sur 20 caractères maximum, de l'adresse de départ et de la longueur de l'enregistrement.

Deux caractères de synchronisation sont utilisés par le programme, 0 pour le Header et #FF pour les Data. Les programmes ainsi sauvés ne seront pas reconnus par les commandes classiques LOAD et SAVE.

REMARQUE: Si lors de la lecture, vous voyez apparaître à l'écran le message "ERREUR DE LECTURE", cela signifie que le système ne parvient plus à se synchroniser. Nous prions donc le lecteur de jouer sur la vitesse d'écriture (routine POSVIT #AllC) pour essayer de trouver la vitesse adéquate. La qualité du lecteur et des cassettes influence grandement le résultat escompté.

Le programme fait appel à six routines internes dont voici les conditions d'entrée et de sortie nécessaires à la compréhension du listing:

ROUTINE : CAS INITIALISE	ADRESSE : BC65
-----	-----

Initialisation du gestionnaire cassette.

Conditions d'entrée : rien
Conditions de sortie : AF, BC, DE et HL sont modifiés.

=====

ROUTINE : CAS SET SPEED

ADRESSE : BC68

Vitesse d'écriture.

Conditions d'entrée : HL contient la valeur d'un demi-zéro.
A contient la précompensation.

Conditions de sortie : AF et HL sont modifiés.

=====

ROUTINE : CAS READ

ADRESSE : BCAl

Lit un enregistrement sur cassette.

Conditions d'entrée : HL contient l'adresse où seront
écrites les données, DE contient le
nombre d'octets à lire et A contient
le caractère de synchronisation.

Conditions de sortie : Le carry est vrai. En cas de mauvaise
lecture, il est faux et A contient un
code d'erreur. De toutes façons, AF,
BC, DE et IX sont modifiés.

=====

ROUTINE : CAS WRITE

ADRESSE : BC9E

Ecrit un enregistrement sur cassette.

Conditions d'entrée : HL contient l'adresse des données à
écrire. DE contient le nombre
d'octets à écrire. A contient le
caractère de synchronisation.

Conditions de sortie : Le carry est vrai. En cas de mauvaise
lecture, il est faux et A contient un
code d'erreur. De toutes façons, HL
et IX sont modifiés.

=====


```

10 MEMORY &9FFF
20 FOR I=&A000 TO &A1DF
30 READ A$
40 POKE i,VAL("&h"+a$)
50 NEXT i
60 CALL &A000
70 DATA 01,0A,A0,21,F8,A1,CD,D1,BC,C9,12,A0,C3,28,A0,C3,80,A
0,53,55,50,45,52,53,41,55,56,C5,53,55,50,45,52,43,48,41,52,4
7,C5,00,FE,01,28,21,FE,03,C2,2E,A1,DD,6E,00,DD,66,01,22,F6,A
1,DD,6E,02,DD,66,03,22,F4,A1,DD,23,DD,23,DD,23,DD
80 DATA 23,18,0C,21,00,C0,22,F4,A1,21,00,40,22,F6,A1,CD,F1,A
0,CD,1C,A1,21,DE,A1,11,1E,00,3E,00,CD,9E,BC,D2,3C,A1,2A,F4,A
1,ED,5B,F6,A1,3E,FF,CD,9E,BC,D2,3C,A1,CD,65,BC,C9,FE,01,28,0
F,FE,02,20,12,3E,00,32,FC,A1,DD,23,DD,23,18,11,3E
90 DATA FF,32,FC,A1,18,0A,FE,00,C2,2E,A1,3E,80,32,FC,A1,CD,F
1,A0,CD,43,A1,21,FD,A1,11,1E,00,3E,00,CD,A1,BC,30,7E,CD,4F,A
1,3A,FC,A1,FE,80,28,10,21,DE,A1,11,FD,A1,06,14,1A,BE,20,16,1
3,23,10,F8,CD,66,A1,2A,13,A2,ED,5B,15,A2,3E,FF,CD
100 DATA A1,BC,30,53,C9,3A,FC,A1,B7,28,C1,21,DB,A1,CD,25,A1,
18,B9,DD,6E,00,DD,66,01,46,23,5E,23,56,C5,21,DE,A1,06,14,36,
20,23,10,FB,C1,78,B7,C8,FE,15,38,02,3E,14,47,21,DE,A1,1A,77,
13,23,10,FA,C9,21,A7,00,3E,32,CD,68,BC,C9,7E,B7,C8
110 DATA CD,5A,BB,23,18,F7,21,8C,A1,CD,25,A1,C9,21,A0,A1,CD,
25,A1,C9,21,C5,A1,CD,25,A1,C9,21,72,A1,3A,FC,A1,B7,C8,CD,25,
A1,C9,21,80,A1,3A,FC,A1,B7,C8,CD,25,A1,3E,00,32,11,A2,21,FD,
A1,CD,25,A1,C9,21,B6,A1,3A,FC,A1,B7,C8,CD,25,A1,C9
120 DATA 0D,0A,52,45,43,48,45,52,43,48,45,0D,0A,00,54,52,4F,
55,56,45,52,3A,20,20,20,00,45,52,52,45,55,52,20,44,45,20,53,
59,4E,54,41,58,45,0D,0A,00,0D,0A,45,52,52,45,55,52,20,44,45,
20,4C,45,43,54,55,52,45,0D,0A,00,0D,0A,43,48,41,52
130 DATA 47,45,4D,45,4E,54,3A,20,00,0D,0A,45,52,52,45,55,52,
20,44,27,45,43,52,49,54,55,52,45,0D,0A,00,0D,0A,00,00,00

```

Vous trouverez à la page suivante le programme source de construction des RSX.

Pass 1 errors: 00

```

10 ;PROGRAMME DE LECTURE ET ECRITURE
20 ;RAPIDE SUR CASSETTE
30 ;PH JADOUL 1985
40 ;
50 ;
A000 60          ORG  #A000
70 ;
80 ;CONSTRUCTION DU RSX
90 ;
A000 010AA0 100          LD  BC,TABCOM
A003 21F8A1 110          LD  HL,TAMPON
A006 CDD1BC 120          CALL #BCD1
A009 C9      130          RET
A00A 12A0    140 TABCOM: DEFW TABLE
A00C C328A0 150          JP   SAVE
A00F C380A0 160          JP   LOAD
A012 53555045 170 TABLE: DEFM "SUPERSAUV"
A01B C5      180          DEFB "E"+#80
A01C 53555045 190          DEFM "SUPERCHARG"
A026 C5      200          DEFB "E"+#80
A027 00      210          DEFB 0
220 ;
230 ;PROG ECRITURE CASSETTE
240 ;
A028 FE01    250 SAVE:   CP    1
A02A 2821    260          JR    Z,SSCREE
270 ;COMPARE LE NB DE PARAMETRES
A02C FE03    280          CP    3
A02E C22EA1  290          JP    NZ,ERROR
300 ;SAUVE LONG ENREG, POS 1 OCTET
A031 DD6E00  310          LD    L,(IX+0)
A034 DD6601  320          LD    H,(IX+1)
A037 22F6A1  330          LD    (LENG),HL
A03A DD6E02  340          LD    L,(IX+2)
A03D DD6603  350          LD    H,(IX+3)
A040 22F4A1  360          LD    (START),HL
A043 DD23    370          INC   IX
A045 DD23    380          INC   IX
A047 DD23    390          INC   IX
A049 DD23    400          INC   IX
A04B 180C    410          JR    LECNOM
420 ;ADD DEBUT MEMOIRE ECRAN
A04D 2100C0  430 SSCREE: LD    HL,#C000
A050 22F4A1  440          LD    (START),HL
450 ;LONG MEMOIRE

```



```

455 *E
A053 210040 460 LD HL,#4000
A056 22F6A1 470 LD (LENG),HL
480 ;LECT DU FICHIER A SAUVER
A059 CDF1A0 490 LECNOM: CALL GETNAM
500 ;POSIT VITESSE ECRITURE
A05C CD1CA1 510 CALL POSVIT
520 ;ECRITURE DU HEADER
A05F 21DEA1 530 LD HL,FNOM
A062 111E00 540 LD DE,30
A065 3E00 550 LD A,0
A067 CD9EBC 560 CALL #BC9E
A06A D23CA1 570 JP NC,OERP
580 ;ECRITURE DES DONNEES
A06D 2AF4A1 590 LD HL,(START,
A070 ED5BF6A1 600 LD DE,(LENG)
A074 3EFF 610 LD A,255
A076 CD9EBC 620 CALL #BC9E
A079 D23CA1 630 JP NC,OERR
A07C CD65BC 640 CALL #BC65
A07F C9 650 RET
660 ;
670 ;
680 ;PROGRAMME LECTURE CASSETTE
690 ;
700 ;COMPARE LE NB DE PARAMETRE
A080 FE01 710 LOAD: CP 1
A082 280F 720 JR Z,PRNOM
A084 FE02 730 CP 2
A086 2012 740 JR NZ,NOPAR
750 ;NMSG=0 PAS D'AFFICH ECRAN
A088 3E00 760 LD A,0
A08A 32FCA1 770 LD (NMSG),A
A08D DD23 780 INC IX
A08F DD23 790 INC IX
A091 1811 800 JR LECT
810 ;LECT AVEC NOM DE FICHIER
A093 3EFF 820 PRNOM: LD A,255
A095 32FCA1 830 LD (NMSG),A
A098 180A 840 JR LECT
850 ;LECT SANS NOM DE FICHIER
A09A FE00 860 NOPAR: CP 0
A09C C22EA1 870 JP NZ,ERROR
A09F 3E80 880 LD A,128
A0A1 32FCA1 890 LD (NMSG),A
900 ;LECT DU NOM DU FICHIER A CHARGER

```


		905	*E
AOA4	CDF1A0	910	LECT: CALL GETNAM
		920	;AFFICH MSG (RECHERCHE)
AOA7	CD43A1	930	CALL RMSG
		940	;LECTURE DU HEADER SUR CASSETTE
AOAA	21FDA1	950	LOOP: LD HL,FNOM1
AOAD	111E00	960	LD DE,30
AOB0	3E00	970	LD A,0
AOB2	CDA1BC	980	CALL #BCA1
AOB5	307E	990	JR NC,IOERR
		1000	;ECRIT DU NOM SUR ECRAN
AOB7	CD4FA1	1010	CALL PRFNOM
Aoba	3AFCA1	1020	LD A,(NMSG)
AOBD	FE80	1030	CP 128
AOBF	2810	1040	JR Z,IDEM
		1050	;COMPARE FNOM AVEC FNOM1
AOC1	21DEA1	1060	LD HL,FNOM
AOC4	11FDA1	1070	LD DE,FNOM1
AOC7	0614	1080	LD B,20
AOC9	1A	1090	CMP: LD A,(DE)
AOCA	BE	1100	CP (HL)
AOCB	2016	1110	JR NZ,NOIDEM
AOCD	13	1120	INC DE
AOCE	23	1130	INC HL
AOCF	10F8	1140	DJNZ CMP
		1150	;SI FNOM=FNOM1 CHARGER DONNEES
AOD1	CD66A1	1160	IDEM: CALL LECDAT
AOD4	2A13A2	1170	LD HL,(START1)
AOD7	ED5B15A2	1180	LD DE,(LENG1)
AODB	3EFF	1190	LD A,255
AODD	CDA1BC	1200	CALL #BCA1
AOE0	3053	1210	JR NC,IOERR
AOE2	C9	1220	RET
		1230	;SI FNOM # FNOM1 RECH AUTRE HEADER
AOE3	3AFCA1	1240	NOIDEM: LD A,(NMSG)
AOE6	B7	1250	OR A
AOE7	28C1	1260	JR Z,LOOP
AOE9	21DBA1	1270	LD HL,CRLF
AOEC	CD25A1	1280	CALL SMSG
AOEF	18B9	1290	JR LOOP
		1300	;
		1310	;
		1320	;ROUTINE
		1330	;
		1340	;ROUTINE LECTURE PARM NOM FICHER
		1350	;LECT ADD DESCRIPTEUR VAR CHAINE

		1355 *E
		1360 ;ET DE SA LONG
A0F1	DD6300	1370 GETNAM: LD L,(IX+0)
A0F4	DD6301	1380 LD H,(IX+1)
A0F7	46	1390 LD B,(HL)
		1400 ;LECT DE ADD DE LA TABLE CHAINE
A0F8	23	1410 INC HL
A0F9	5E	1420 LD E,(HL)
A0FA	23	1430 INC HL
A0FB	56	1440 LD D,(HL)
		1450 ;MISE A BLANC D'UNE ZONE DE 20 CAR
A0FC	C5	1460 PUSH BC
A0FD	21DEA1	1470 LD HL,FNOM
A100	0614	1480 LD B,20
A102	3620	1490 BLKNOM: LD (HL),32
A104	23	1500 INC HL
A105	10FB	1510 DJNZ BLKNOM
A107	C1	1520 POP BC
A108	78	1530 LD A,B
A109	B7	1540 OR A
A10A	C8	1550 RET Z
		1560 ;SI B=21 LONG NOM OK
A10B	FE15	1570 CP 21
A10D	3802	1580 JR C, LONGOK
A10F	3E14	1590 LD A,20
A111	47	1600 LONGOK: LD B,A
A112	21DEA1	1610 LD HL,FNOM
A115	1A	1620 COPNOM: LD A,(DE)
A116	77	1630 LD (HL),A
A117	13	1640 INC DE
A118	23	1650 INC HL
A119	10FA	1660 DJNZ COPNOM
A11B	C9	1670 RET
		1680 ;
		1690 ; POSIT VITESSE ECRITURE
		1700 ;
		1710 ;HL VALEUR DEMI ZERO, A PRECOMP
A11C	21A700	1720 POSVIT: LD HL,167
A11F	3E32	1730 LD A,50
A121	CD68BC	1740 CALL #BC68
A124	C9	1750 RET
		1760 ;
		1770 ;ROUT D'IMPRESSION MSG ECRAN
		1780 ;
A125	7E	1790 SMSG: LD A,(HL)
A126	B7	1800 OR A

		1805 *E	
A127	C8	1810	RET Z
A128	CD5ABB	1820	CALL #BB5A
A12B	23	1830	INC HL
A12C	18F7	1840	JR SMSG
		1850 ;	
		1860 ;	ROUT APPEL DES MSG
		1870 ;	
A12E	218CA1	1880	ERROR: LD HL,MSG3
A131	CD25A1	1890	CALL SMSG
A134	C9	1900	RET
		1910 ;	
A135	21A0A1	1920	IOERR: LD HL,MSG4
A138	CD25A1	1930	CALL SMSG
A13B	C9	1940	RET
		1950 ;	
A13C	21C5A1	1960	OERR: LD HL,MSG6
A13F	CD25A1	1970	CALL SMSG
A142	C9	1980	RET
		1990 ;	
A143	2172A1	2000	RMSG: LD HL,MSG1
A146	3AFCA1	2010	LD A,(NMSG)
A149	B7	2020	OR A
A14A	C8	2030	RET Z
A14B	CD25A1	2040	CALL SMSG
A14E	C9	2050	RET
		2060 ;	
A14F	2180A1	2070	PRFNOM: LD HL,MSG2
A152	3AFCA1	2080	LD A,(NMSG)
A155	B7	2090	OR A
A156	C8	2100	RET Z
A157	CD25A1	2110	CALL SMSG
A15A	3E00	2120	LD A,0
A15C	3211A2	2130	LD (FNOM1+20),A
A15F	21FDA1	2140	LD HL,FNOM1
A162	CD25A1	2150	CALL SMSG
A165	C9	2160	RET
		2170 ;	
A166	21B6A1	2180	LECDAT: LD HL,MSG5
A169	3AFCA1	2190	LD A,(NMSG)
A16C	B7	2200	OR A
A16D	C8	2210	RET Z
A16E	CD25A1	2220	CALL SMSG
A171	C9	2230	RET
		2240 ;	
		2250 ;	


```

2255 *E
2260 ;DEFINITIONS DES MSG
2270 ;
A172 0D0A 2280 MSG1: DEFB 13,10
A174 52454348 2290 DEFM "RECHERCHE"
A17D 0D0A00 2300 DEFB 13,10,0
2310 ;
A180 54524F55 2320 MSG2: DEFM "TROUVER"
A187 3A202020 2330 DEFB 58,32,32,32,0
A18C 45525245 2340 MSG3: DEFM "ERREUR DE SYNTAXE"
A19D 0D0A00 2350 DEFB 13,10,0
2360 ;
A1A0 0D0A 2370 MSG4: DEFB 13,10
A1A2 45525245 2380 DEFM "ERREUR DE LECTURE"
A1B3 0D0A00 2390 DEFB 13,10,0
2400 ;
A1B6 0D0A 2410 MSG5: DEFB 13,10
A1B8 43484152 2420 DEFM "CHARGEMENT"
A1C2 3A2000 2430 DEFB 58,32,0
2440 ;
A1C5 0D0A 2450 MSG6: DEFB 13,10
A1C7 45525245 2460 DEFM "ERREUR D'ECRITURE"
A1D8 0D0A00 2470 DEFB 13,10,0
2480 ;
2490
A1DB 0D0A00 2500 CRLF: DEFB 13,10,0
2510 ;
2520 ;
2530 ;DEBUT DU HEADER BLOC1
2540 ;
A1DE 2550 FNOM: DEFS 22
A1F4 2560 START: DEFS 2
A1F6 2570 LENG: DEFS 2
A1F8 2580 TAMPON: DEFS 4
A1FC 2590 NMSG: DEFS 1
2600 ;
2610 ;DEBUT DU HEADER BLOC2
2620 ;
A1FD 2630 FNOM1: DEFS 22
A213 2640 START1: DEFS 2
A215 2650 LENG1: DEFS 2
2660

```


10.13 RSX HARDCOPY.

Le programme de HARDCOPY, ou copie écran, que nous allons décrire fait appel aussi bien aux routines du gestionnaire imprimante qu'à celles des gestionnaires écran et clavier.

DESCRIPTION.

L'écran est composé de 25 lignes dont le nombre de caractères par ligne dépend du mode utilisé: 20 pour le mode 0, 40 pour le mode 1 et 80 pour le mode 2.

=====

ROUTINE : SCR GET MODE

ADRESSE : BC11

Routine du gestionnaire écran qui nous renseigne sur le mode utilisé

Conditions d'entrée : rien

Conditions de sortie : A contient le numéro du mode, le carry et le zéro sont positionnées en fonction de celui-ci:
MODE 0 : C=1 Z=0, A=0
MODE 1 : C=0 Z=1, A=1
MODE 2 : C=0 Z=0, A=2

=====

Lors de l'écriture du programme, nous utiliserons la réponse à cette routine pour définir un facteur multiplicateur du nombre de caractères à imprimer par ligne.

Si l'accumulateur contient la valeur 0, l'écran est en mode 0. En exécutant sur l'accumulateur l'instruction RLA, qui signifie rotation à gauche avec report du carry dans le bit B0 de l'accumulateur, nous obtenons dans A la valeur 1 :

		registre A							
		B7							B0
C=1		0	0	0	0	0	0	0	0

RLA

		registre A							
		B7							B0
C=0		0	0	0	0	0	0	0	1

qui est le facteur de multiplication pour le mode 0, c'est-à-dire, 1 fois 20 caractères = 20.

Pour le mode 1:

		registre A							
		B7							B0
C=0		0	0	0	0	0	0	0	1

RLA

		registre A							
		B7							B0
C=0		0	0	0	0	0	0	1	0

A=2. Le facteur est donc égal à 2. 2 X 20 caractères = 40

Pour le mode 2:

		registre A							
		B7							B0
C=0		0	0	0	0	0	0	1	0

RLA

		registre A							
		B7							B0
C=0		0	0	0	0	0	1	0	0

A=4. Le facteur est égal à 4. 4 X 20 = 80 caractères.

Le programme de HARDCOPY ayant pour but de balayer toute la mémoire écran en envoyant sur l'imprimante tous les caractères rencontrés, la première opération à effectuer est de sauvegarder la position actuelle du curseur pour pouvoir la restituer à la fin du programme.

Cette tâche est rendue possible grâce à la routine suivante:

=====

ROUTINE : TXT GET CURSOR	ADRESSE : BB78
-----	-----

Routine du gestionnaire écran qui permet la lecture de la position du curseur.

Conditions d'entrée : rien.
Conditions de sortie : H contient le numéro de la colonne du curseur et L, le numéro de la ligne. A contient le compteur de défilement (scrolling).

=====

La seconde opération consiste à positionner le curseur en début d'écran (coin supérieur gauche):

=====

ROUTINE : TXT SET CURSOR	ADRESSE : BB75
-----	-----

Positionnement du curseur.

Conditions d'entrée : H contient le numéro de la colonne du curseur et L, le numéro de la ligne.
Conditions de sortie : AF et HL sont modifiés.

=====

La troisième opération consiste à extraire un caractère de l'écran:

=====

ROUTINE : TXT RD CHAR	ADRESSE : BB60
-----	-----

Routine du gestionnaire écran qui permet la lecture d'un caractère en provenance de l'écran à la position courante du curseur.

Conditions d'entrée : rien.
Conditions de sortie : Si le caractère a été reconnu, le sémaphore de carry est vrai et A contient le caractère.

=====

PROGRAMME BASIC D'INITIALISATION DU RSX HARDCOPY.

```
10 MEMORY &9FFF
20 FOR i=&A000 TO &A083
30 READ a$
40 POKE i,VAL("&h"+a$)
50 NEXT i
60 CALL &A000
70 DATA 00,00,00,00,00,FC,A6,13,A0,01,13,A0,21,05,A0,CD,D1,B
C,C9,18,A0,C3,1E,A0,48,43,4F,50,D9,00,CD,78,BB,22,01,A0,CD,1
1,BC,17,32,00,A0,21,01,01,22,03,A0,3A,00,A0,47,0E,14,2A,03,A
0,CD,75,BB,CD,60,BB,38,02,3E,20,CD,7B,A0,C5,3E,42
80 DATA CD,1E,BB,38,25,C1,2A,03,A0,24,22,03,A0,0D,20,DD,10,D
9,3E,0D,CD,7B,A0,3E,0A,CD,7B,A0,2A,03,A0,26,01,2C,22,03,A0,7
D,FE,1A,20,BD,2A,01,A0,CD,75,BB,C9,CD,2E,BD,38,FB,CD,31,BD,C
9
```

SYNTAXE : :HCOPY

Vous trouverez, à la page suivante, le programme source du programme de HARDCOPY.

Pass 1 errors: 00

A000		10	ORG #A000
BB1E		20	TCLAV: EQU #BB1E
		30	;ROUT TEST D'UNE TOUCHE
BB60		40	CARCRT: EQU #BB60
		50	;ROUT PRISE D'UN CAR
BB75		60	PCURS: EQU #BB75
		70	;ROUT POS CURSOR
BB78		80	LCURS: EQU #BB78
		90	;ROUT LECT POS CURSOR
BC11		100	LMODE: EQU #BC11
		110	;LECT MODE ECRAN
BD2E		120	TIMP: EQU #BD2E
		130	;TEST IMPR
BD31		140	CARIMP: EQU #BD31
		150	;ENVOIE CAR A IMPR
A000	00	160	MODE: DEFB 0
A001	0000	170	ANCPOS: DEFW 0000
A003	0000	180	CRTPOS: DEFW 0000
A005		190	TAMPON: DEFS 4
		200	;
		210	;INTRODUCTION DU RSX
		220	;
A009	0113A0	230	LD BC,TABCOM
A00C	2105A0	240	LD HL,TAMPON
A00F	CDD1BC	250	CALL #BCD1
A012	C9	260	RET
A013	18A0	270	TABCOM: DEFW TABLE
A015	C31EA0	280	JP HCOPY
A018	48434F50	290	TABLE: DEFM "HCOP"
A01C	D9	300	DEFB "Y"+#80
A01D	00	310	DEFB 0
		320	;
		330	;
		340	;PROG DU HARDCOPY
		350	;
A01E	CD78BB	360	HCOPY: CALL LCURS
A021	2201A0	370	LD (ANCPOS),HL
A024	CD11BC	380	CALL LMODE
A027	17	390	RLA
A028	3200A0	400	LD (MODE),A
		410	;
		420	;POSI CURSOR EN HAUT
		430	;ET AGAUCHE DE L'ECRAN
A02B	210101	440	LD HL,#0101
A02E	2203A0	450	LD (CRTPOS),HL

		455	*e	
A031	3A00A0	460	LOOP:	LD A,(MODE)
A034	47	470		LD B,A
A035	0E14	480	LOOP1:	LD C,#14
		490	;	
		500	;	LECT POS ECRAN
A037	2A03A0	510	LOOP2:	LD HL,(CRTPOS)
A03A	CD75BB	520		CALL PCURS
		530	;	
		540	;	LECT D'UN CAR
A03D	CD60BB	550		CALL CARCRT
A040	3802	560		JR C,OK
		570	;	
		580	;	SI LE CAR NON VALABLE
		590	;	IMPRIMER UN BLANC
A042	3E20	600		LD A,#20
		610	;	IMP D'UN CAR.
A044	CD7BA0	620	OK:	CALL SPIMPR
A047	C5	630		PUSH BC
		640	;	
		650	;	TEST DE LA TOUCHE ESC
A048	3E42	660		LD A,66
A04A	CD1EBB	670		CALL TCLAV
A04D	3825	680		JR C,FIN
A04F	C1	690		POP BC
		700	;	
		710	;	POS CAR ECRAN SUIVANT
A050	2A03A0	720		LD HL,(CRTPOS)
A053	24	730		INC H
A054	2203A0	740		LD (CRTPOS),HL
A057	0D	750		DEC C
A058	20DD	760		JR NZ,LOOP2
A05A	10D9	770		DJNZ LOOP1
		780	;	
		790	;	EN FIN DE LIGNE CR ET LF
A05C	3E0D	800		LD A,#0D
A05E	CD7BA0	810		CALL SPIMPR
A061	3E0A	820		LD A,#0A
A063	CD7BA0	830		CALL SPIMPR
		840	;	
		850	;	POS CURSOR SUR LE PREMIER
		860	;	CAR DE LA LIGNE SUIVANTE
A066	2A03A0	870		LD HL,(CRTPOS)
A069	2601	880		LD H,#01
A06B	2C	890		INC L
A06C	2203A0	900		LD (CRTPOS),HL


```

          905 *e
          910 ;
          920 ;TEST DE FIN D'ECRAN
A06F 7D          930          LD    A,L
A070 FE1A        940          CP    26
A072 20BD        950          JR    NZ,LOOP
          960 ;
          970 ;REST DE LA POS CURSOR
A074 2A01A0      980 FIN:      LD    HL,(ANCPOS)
A077 CD75BB      990          CALL PCURS
A07A C9          1000         RET
          1010 ;
A07B            1020 ;
          1030 ;ROUT IMPRES D'UN CAR
A07B CD2EBD      1040 SPIMPR: CALL TIMP
A07E 38FB        1050          JR    C,SPIMPR
A080 CD31BD      1060          CALL CARIMPR
A083 C9          1070         RET

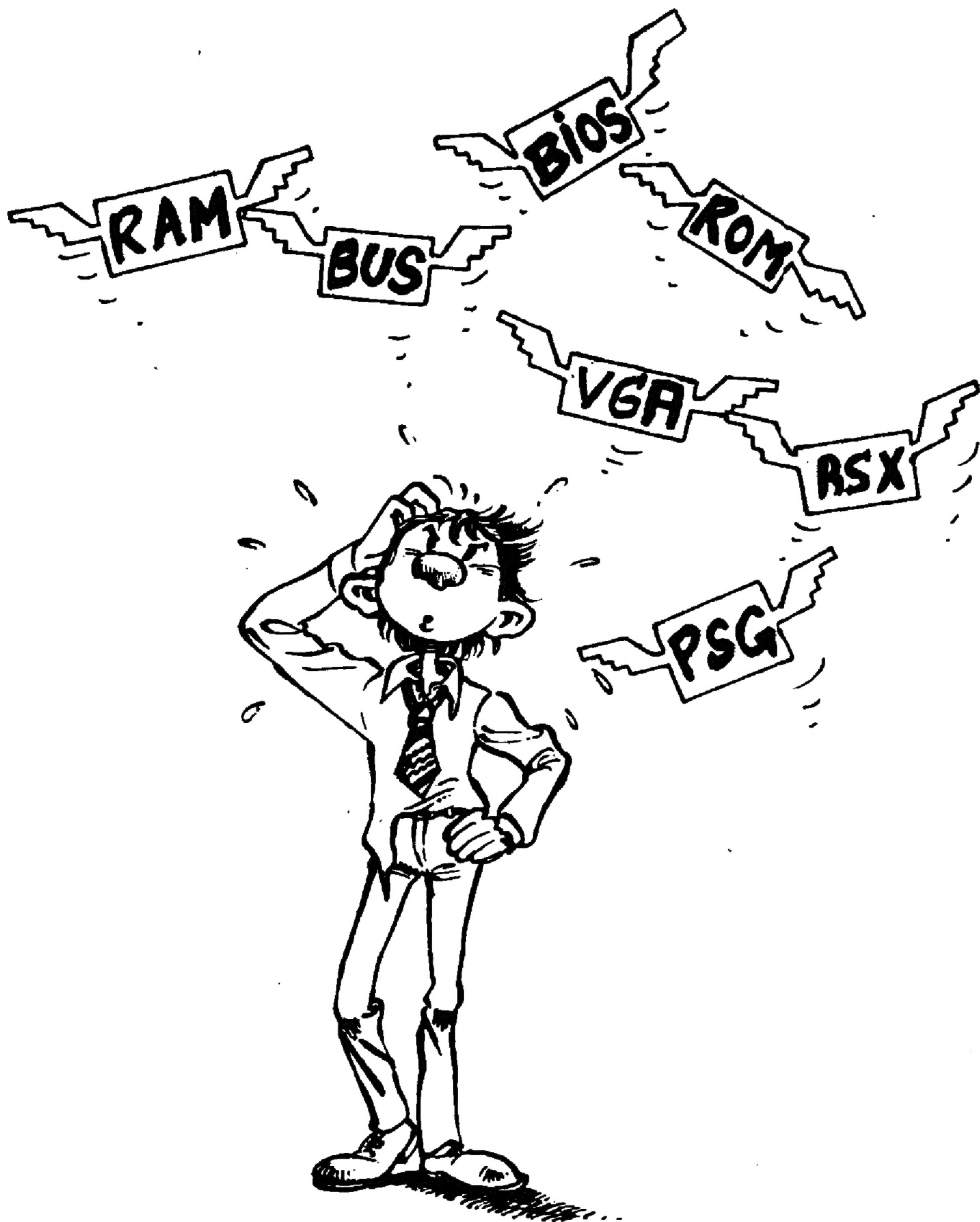
```

Pass 2 errors: 00

ANCPOS	A001	CARCRT	BB60	CARIMP	BD31
CRTPOS	A003	FIN	A074	HCOPY	A01E
LCURS	BB78	LMODE	BC11	LOOP	A031
LOOP1	A035	LOOP2	A037	MODE	A000
OK	A044	PCURS	BB75	SPIMPR	A07B
TABCOM	A013	TABLE	A018	TAMPON	A005
TCLAV	BB1E	TIMP	BD2E		

Table used: 252 from 1000

Notes.



B I B L I O G R A P H I E

Ouvrages en français.

CLEFS POUR AMSTRAD de Daniel Martin (PSI)

LE LIVRE DU MSX de Daniel Martin (BCM-PSI)

LA BIBLE DU PROGRAMMEUR de Micro application (DATABECKER)

PROGRAMMATION DU Z80 par Rodney Zaks (SYBEX)

L'ASSEMBLEUR DE L'AMSTRAD par Marcel Henrot (PSI)

Ouvrages en anglais.

The COMPLETE CFC464 Operating System firmware Specifications

CRT Controller Handbook by Gerry Kane (OSBORNE)

AMSTRAD ASSEMBLY LANGUAGE COURSE by Tim Herbertson (WATSON)

The INS & OUTS of the AMSTRAD by Don Thomson (MELBOURNE)

THE AMSTRAD PROGRAMMERS GUIDE by Bryan Skinner (DUCKWORTH)

PSG Microelectronics data catalog (General Instrument)

Intel data book microprocessor (PPI 8255)

BASIC Faster and Better by Lewis Rosenfelder (IJG).

Imprimé en Belgique par BON TON - 4600 CHENEE

Dépôt légal : D/1985/3827/6

LE LIVRE DE L'AMSTRAD

« LE LIVRE DE L'AMSTRAD »

dévoile au lecteur la «face cachée» de son ordinateur.

Etude complète de tous les circuits internes de l'Amstrad, analyse de la structure interne du Basic.

Etude poussée des fonctions et instructions mal connues du Basic dont la fonction VARPTR génératrice de prodiges et pourtant ignorée de la plupart des manuels.

Etude complète des RSX, ces merveilleux outils permettant d'ajouter de nouvelles commandes au Basic.

Nombreux programmes permettant d'ajouter au Basic les commandes de scrolling, de traçage de rectangles et de cercles, de coloriage de surface et de manipulation vectorielle.

Liste des différences entre les CPC 464 et 664 permettant ainsi d'adapter tous les programmes d'une machine vers l'autre.

En bref, cet ouvrage est destiné aux utilisateurs de CPC 464, 664 et 6128 désireux de connaître à fond leur ordinateur et d'en utiliser efficacement les ressources.

BCM s.c.

24, route de la Sapinière - 4960 Banneux Belgique



9 782871 110057

ISBN 2-87111-005-0



120 FF